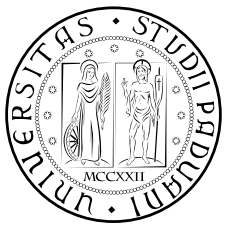


UNIVERSITÀ DEGLI STUDI DI PADOVA

FACOLTÀ DI INGEGNERIA
CORSO DI LAUREA MAGISTRALE IN INGEGNERIA MECCATRONICA

TESI DI LAUREA MAGISTRALE

SVILUPPO DI UNA RETE DI SENSORI WIRELESS PER IL MONITORAGGIO DI COLATE DETRITICHE TORRENTIZIE

Relatore: Reggiani Monica

Correlatore: Cenedese Angelo

Laureando: Marogna Simone
620778-IMC

ANNO ACCADEMICO: 2011-12

RINGRAZIAMENTI

Desidero inanzitutto ringraziare la Professoressa Monica Reggiani che si è sempre resa disponibile durante la preparazione della tesi rendendo questo momento meno carico d'ansia. Inoltre ringrazio sentitamente il Professor Angelo Cenedese ed il suo gruppo di ricerca al DEI, sottolineando la particolare disponibilità di Felicini Nicholas che mi ha più volte aiutato nella soluzione dei problemi incontrati durante lo sviluppo del progetto.

Un grazie di cuore per l'affetto che mi hanno sempre saputo dare a tutti i miei famigliari; in modo particolare ai miei genitori, vi voglio bene e se sono arrivato fin qui lo devo a voi.

INDICE

1 INTRODUZIONE 1
- 1.1 Le colate detritiche torrentizie 1
  - 1.1.1 Prevenzione e sistemi di allerta 4
- 1.2 Analisi del problema 6
- 1.3 Schema della tesi 7

2 SOLUZIONE AL PROBLEMA DELLA LOCALIZZAZIONE 9
- 2.1 Algoritmo di trilaterazione 9
  - 2.1.1 Metodo di Newton 10
- 2.2 Implementazione 11

3 ANALISI IN SIMULAZIONE 17
- 3.1 Configurazioni statiche 20
  - 3.1.1 Configurazione 1: disposizione ordinata dei mote con 3 ancòre 21
  - 3.1.2 Configurazione 2: disposizione ordinata dei mote con 4 ancòre 23
  - 3.1.3 Configurazione 3: disposizione casuale dei mote con 3 ancòre 29
  - 3.1.4 Conclusioni sulla valutazione sperimentale nel caso statico 32
- 3.2 Configurazioni dinamiche 33
  - 3.2.1 Configurazione 1: Spostamento del nodo 6 con 10 passi da 5cm 33
  - 3.2.2 Configurazione 1: Spostamento del nodo 6 con 5 passi da 10cm 36
  - 3.2.3 Conclusioni sulla valutazione sperimentale nel caso dinamico 38
- 3.3 Conclusioni della valutazione sperimentale 39

4 REALIZZAZIONE SU DISPOSITIVI TMOTE SKY 41
- 4.1 Tmote Sky 41
  - 4.1.1 Alimentazione 42
  - 4.1.2 Microcontrollore 42
  - 4.1.3 Comunicazione con il PC e programmazione 44
  - 4.1.4 Flash esterna 45
  - 4.1.5 Connettore di espansione 45
  - 4.1.6 Radio 45
- 4.2 Sistema TinyOS 47
  - 4.2.1 Componenti 48
  - 4.2.2 Split-phase operation 50
  - 4.2.3 Active messages 51
- 4.3 Linguaggio nesC 52
  - 4.3.1 Moduli e configurazioni 53
  - 4.3.2 Assemblaggio dei componenti 55

4.3.3 Stratificazione dei componenti 55
4.3.4 Modello concorrente 56
4.4 Modello del canale 57
4.4.1 Calcolo del modello del canale 59
4.4.2 Analisi del modello 60
4.4.3 Valutazione algoritmo di trilaterazione con dati reali 63
4.5 Accelerometro 67
4.5.1 Teoria del funzionamento 68
4.5.2 Informazioni sull'utilizzo 71
4.6 Verifica funzionamento accelerometro 72
Conclusioni 75

Appendix 77
A SETUP DEL SOFTWARE 79
A.1 Pacchetti Ubuntu 79
A.1.1 Installazione TinyOS 80
A.1.2 Verifica dell'installazione 80

BIBLIOGRAFIA 83

ELENCO DELLE FIGURE

Figura 1 Esempio della diversa granulometria dei detriti trasportati da un *debris flow*. 1

Figura 2 Esempio di tre ondate successive di un *debris flow* che ha colpito Virgen (Austria) nell'agosto 2012. 2

Figura 3 Ripresa aerea della situazione di Vargas dopo il *debris flow* del 1999 3

Figura 4 Foto del *debris flow* che ha colpito Cancia nel 2009 3

Figura 5 Foto scattata da un abitante di Brentino Belluno durante il *debris flow* del 2009 4

Figura 6 Installazione di una gabbia in località Rosazza, pochi chilometri a nord di Biella. 5

Figura 7 Generico grafo rappresentante una distrubuzione dei nodi. 9

Figura 8 Esempio di vettore didascalia e vettore di stato associato 15

Figura 9 Posizione dei nodi nelle prime due configurazioni. 17

Figura 10 Posizione dei nodi per la configurazione 3. 18

Figura 11 Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 5$. 21

Figura 12 Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 10cm$ 22

Figura 13 Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 20cm$ 23

Figura 14 Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 40cm$ 24

Figura 15 Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 80cm$ 24

Figura 16 Esempio per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 5cm$ 25

Figura 17 Esempio per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 10cm$ 26

Figura 18 Esempio per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 20cm$ 26

Figura 19 Risultato della prova con condizioni iniziali e distanze del caso esempio di 3.1.1.3 ma con l'aggiunta di un ancòra. 27

Figura 20 Esempio per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 40cm$ 28

Figura 21 Situazione riassuntiva per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 80cm$ 28
Figura 22 Esempio per uno degli N test con 3 ancòre, disposizione random e $\delta = 5cm$ 29
Figura 23 Esempio per uno degli N test con 3 ancòre, disposizione random e $\delta = 10cm$ 30
Figura 24 Situazione riassuntiva per uno degli N test con 3 ancòre, disposizione random e $\delta = 20cm$ 31
Figura 25 Situazione riassuntiva per uno degli N test con 3 ancòre, disposizione random e $\delta = 40cm$ 31
Figura 26 Situazione riassuntiva per uno degli N test con 3 ancòre, disposizione random e $\delta = 80cm$ 32
Figura 27 Evoluzione del nodo mobile con 10 passi da 5cm e $\delta = 5cm$ 34
Figura 28 Evoluzione complessiva con 10 passe da 5cm e $\delta = 5cm$ 35
Figura 29 Evoluzione del nodo mobile con 10 passi da 5cm e $\delta = 10cm$ 36
Figura 30 Evoluzione complessiva con 10 passi da 5cm e $\delta = 10cm$ 36
Figura 31 Evoluzione del nodo mobile con 5 passi da 10cm e $\delta = 5cm$ 37
Figura 32 Evoluzione complessiva con 5 passi da 10cm e $\delta = 5cm$ 37
Figura 33 Evoluzione del nodo mobile con 5 passi da 10cm e $\delta = 10cm$ 38
Figura 34 Evoluzione complessiva con 5 passi da 10cm e $\delta = 10cm$ 39
Figura 35 Piattaforma Tmote Sky [8] 43
Figura 36 Diagramma a blocchi funzionali del modulo Tmote Sky. 44
Figura 37 Mappatura dell'indice RSSI in funzione della potenza del segnale ricevuto [8]. 46
Figura 38 Diagramma di radiazione dell'antenna ad F-invertita del Tmote Sky [8]. 47
Figura 39 Architettura a componenti di un'applicazione TinyOS 49
Figura 40 Semantica split-phase. 51
Figura 41 Esempio di grafo dei componenti 55
Figura 42 Stratificazione di un applicazione 56
Figura 43 Grafico dei valori di potenza misurati dal nodo 0 rispetto al nodo 1 61
Figura 44 Grafico dei valori di potenza misurati dal nodo 0 rispetto al nodo 2 62
Figura 45 Grafico dei valori di potenza misurati dal nodo 1 rispetto al nodo 0 63

Figura 46 Grafico dei valori di potenza misurati dal nodo 1 rispetto al nodo 2 63
Figura 47 Grafico dei valori di potenza misurati dal nodo 2 rispetto al nodo 0 64
Figura 48 Grafico dei valori di potenza misurati dal nodo 2 rispetto al nodo 1 65
Figura 49 Stima delle posizioni utilizzando i valori di potenza reali. 66
Figura 50 Stima delle posizioni con i valori di potenza utilizzati in [14]. 66
Figura 51 Valutazione del modello del canale con i dati di [14]. 67
Figura 52 Modello dell'accelerometro 68
Figura 53 Risposta d'uscita vs. orientamento alla gravità [3]. 69
Figura 54 Diagramma a blocchi funzionali dell'accelerometro [3]. 69
Figura 55 Screenshot programma java per visualizzare i valori misurati dall'accelerometro. 73
Figura 56 *Repository source file* 79
Figura 57 File bashrc aggiornato 81

ELENCO DELLE TABELLE

Tabella 1 Coordinate dei mote nelle configurazioni 1 e 2. 19
Tabella 2 Coordinate dei mote nella configurazione 3. 19
Tabella 3 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 5cm$ 21
Tabella 4 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 10cm$ 22
Tabella 5 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 20cm$ 23
Tabella 6 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 40cm$ 23
Tabella 7 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 80cm$ 24

Tabella 8 Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 5cm$ 25
Tabella 9 Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 10cm$ 25
Tabella 10 Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 20cm$ 26
Tabella 11 Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 40cm$ 27
Tabella 12 Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 80cm$ 27
Tabella 13 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 5cm$ 29
Tabella 14 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 10cm$ 30
Tabella 15 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 20cm$ 30
Tabella 16 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 40cm$ 31
Tabella 17 Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 80cm$ 31
Tabella 18 Tabella riassuntiva con i valori di errore medio e deviazione standard per le tre configurazioni. 32
Tabella 19 Media e deviazione standard dell'errore medio di stima per il nodo 6 con spostamento di $50cm$ in 10 passi e $\delta = 5cm$. 34
Tabella 20 Media e deviazione standard dell'errore di stima per il nodo 6 nei tre passi valutati con $\delta = 5cm$ 34
Tabella 21 Media e deviazione standard dell'errore medio di stima per il nodo 6 con spostamento di $50cm$ in 10passi e $\delta = 10cm$ 35
Tabella 22 Media e deviazione standard dell'errore di stima per il nodo 6 nei tre passi valutati con $\delta = 10cm$ 35

Tabella 23 Media e deviazione standard dell'errore medio di stima per il nodo 6 con spostamento di 50cm in 5passi e $\delta = 5$cm 37
Tabella 24 Media e deviazione standard dell'errore di stima per il nodo 6 nei tre passi valutati con $\delta = 5$cm 37
Tabella 25 Media e deviazione standard dell'errore medio di stima per il nodo 6 con spostamento di 50cm in 5passi e $\delta = 10$cm 38
Tabella 26 Media e deviazione standard dell'errore di stima per il nodo 6 nei tre passi valutati con $\delta = 10$cm 38
Tabella 27 Tabella riassuntiva con i valori di errore medio e deviazione standard per le due prove. 39
Tabella 28 Condizioni tipiche di funzionamento della piattaforma Tmote Sky 43
Tabella 29 Configurazioni della potenza di uscita per il CC2420 46
Tabella 30 Tabella relativa alle misure del nodo 0 rispetto al nodo 1 61
Tabella 31 Tabella relativa alle misure del nodo 0 rispetto al nodo 2 62
Tabella 32 Tabella relativa alle misure del nodo 1 rispetto al nodo 0 62
Tabella 33 Tabella relativa alle misure del nodo 1 rispetto al nodo 2 64
Tabella 34 Tabella relativa alle misure del nodo 2 rispetto al nodo 0 64
Tabella 35 Tabella relativa alle misure del nodo 2 rispetto al nodo 1 65
Tabella 36 Tabella relativa alle misure di [14]. 67
Tabella 37 Specifiche dell'accelerometro ADXL335 [3] 70

1 INTRODUZIONE

Lo scopo di questo lavoro è quello di studiare e sviluppare un sistema alternativo e più efficiente a quelli già impiegati per il monitoraggio di colate detritiche torrentizie.

1.1 LE COLATE DETRITICHE TORRENTIZIE

Le colate detritiche torrentizie (in inglese *debris flow*) sono una particolare tipologia di fenomeno franoso e consistono in un processo naturale rappresentato dal trasporto di materiale solido da parte di un fluido in ambiente montano.

I *debris flow* sono colate con elevata concentrazione di materiale detritico, compresa tra il 25 e il 70% del volume totale, che si muove verso valle con velocità variabile da pochi cm/s sino a decine di m/s. Valori tipici che si trovano spesso in letteratura vanno da 25cm/s a 44m/s. Il materiale trasportato ha granulometria molto variabile (Figura 1) ed un singolo fenomeno si manifesta con ondate successive (Figura 2) dovute a temporanee ostruzioni del canale di trasporto.

Figura 1: Esempio della diversa granulometria dei detriti trasportati da un *debris flow*.

Le colate detritiche sono fenomeni diffusi nella maggior parte delle fasce climatiche, dalle regioni desertiche a quelle alpine, e rivestono una notevole importanza sia per la loro influenza sull'evoluzione

(a) Prima ondata

(b) Seconda ondata

(c) Terza ondata

Figura 2: Esempio di tre ondate successive di un *debris flow* che ha colpito Virgen (Austria) nell'agosto 2012.

morfologica dei bacini idrografici nei quali avvengono, sia per il rischio potenziale che determinano sui conoidi alluvionali. In queste aree aumenti improvvisi di disponibilità idrica, dovuti comunemente a piogge intense o alla rapida fusione di nevai, possono provocare, con lo scorrimento dell'acqua lungo i pendii, la mobilitazione di ingenti quantità di detriti che vengono incorporate in un *debris flow*. Frequentemente questi fenomeni, specie quando coinvolgomo masse di detriti modeste, tendono ad esaurirsi sul posto, ma in particolari condizioni morfologiche queste miscele semiliquide si incanalano e divengono elementi destabilizzanti per l'alveo torrentizio.

Questo tipo di fenomeno, che può essere visto in maniera semplificata come una via di mezzo tra una frana ed una piena, è altamente imprevedibile nello spazio e nel tempo. Nello spazio perché fenomeni di colata ricorrenti nella stessa località possono deviare dal percorso usuale o perché un canale di colata si può originare in luoghi mai minacciati prima a memoria d'uomo Nel tempo perché in ambiente montano non è sempre possibile individuare le celle temporalesche (200 m di lato) che usualmente producono deflusso superficiale atto a generare colate di detrito.

Le colate detritiche consistono in miscugli di materiale fine (sabbia, limo e argilla) e grossolano (ghiaia e massi), contenenti una quantità variabile di acqua, cui si associano spesso tronchi d'albero ed altri detriti vegetali. Si forma così una massa fangosa in sospensione acquosa che si propaga lungo le aste torrentizie come un unico corpo, senza separazione tra la fase solida e quella liquida. Si tratta quindi di un fluido non newtoniano[1], ciò detemina l'elevatissima capacità

1 Si definisce non newtoniano un fluido la cui viscosità varia a seconda dello sforzo di taglio che viene applicato

erosiva propria di questi fenomeni. Inoltre i moti convettivi che si sviluppano all'interno della massa tengono in superficie i massi di maggiori dimensioni o i tronchi d'albero raccolti lungo il percorso e che vengono a concentrarsi sulla sommità del deposito formando dei gradienti inversi di distribuzione granulometrica. Queste caratteristiche portano il fenomeno dei *debris flow* ad avere un consistente effetto, sia in termini di erosione fondale che spondale. In questo secondo caso le conseguenze sono più gravi in quanto l'indebolimento di tratti di versante porta al crollo di ulteriori consistenti masse di materiale, inoltre può portare allo sviluppo di ulteriori fenomeni di innesco, ed è questo il motivo per cui i *debris flow* si presentano ad ondate successive durante le quali è possibile che si verifichi la rimobilitazione del materiale precedentemente depositato.

La capacità distruttiva di questi fenomeni può raggiungere livelli catastrofici, come nel caso di Vargas, cittadina del Venezuela, nella quale il 15 dicembre 1999 un *debris flow* ha causato la morte di decine di migliaia di persone e distrutto 9000 abitazioni [4]. In Figura 3 è riportata una ripresa aerea della situazione della cittadina dopo l'evento.

Esempi meno clamorosi ma più vicini alla nostra realtà territoriale sono quelli di Cancia, vicino a Borca di Cadore (Belluno), del luglio 2009 che ha causato 2 morti e 300 sfollati (Figura 4); e Brentino Belluno (Verona), che nel dicembre 2009 è stato letteralmente sommerso da più di mille metri cubi di fango e detriti (Figura 5).

Figura 3: Ripresa aerea della situazione di Vargas dopo il *debris flow* del 1999

Figura 4: Foto del *debris flow* che ha colpito Cancia nel 2009

Figura 5: Foto scattata da un abitante di Brentino Belluno durante il *debris flow* del 2009

1.1.1 *Prevenzione e sistemi di allerta*

A causa dell'elevata capacità distruttiva di questo fenomeno risulta conveniente monitorarne l'attività allo scopo di limitare perdite umane, danni a centri abitati e vie di comunicazione allarmando tempestivamente le autorità competenti. La limitazione dei danni può essere fatta agendo sulle cause che generano un *debris flow* o ostacolandone la propagazione quando è già esistente. Si parla quindi di prevenzione attiva e prevenzione passiva.
La prevenzione attiva prevede di intervenire sui possibili alvei con opere di sanamento, con una regimazione completa dei canali e il rafforzamento di tutte le strutture di contenimento dei versanti. Tutti questi interventi comportano elevate spese richiedendo quindi ingenti finanziamenti pubblici, per questo motivo spesso si preferisce ricorrere alla prevenzione passiva.
La prevenzione passiva consiste nell'installazione di opere di difesa che interferendo con una eventuale colata detritica ne impediscano la propagazione; la tecnologia che sta riscuotendo maggior successo è quella basata sull'installazione di briglie, paragonabili a quelle utilizzate come paramassi, costituite da reti in acciaio che uniscono alla capacità di assorbire elevatissime energie di impatto una completa trasparenza al flusso idraulico di superficie (Figura 6). Queste sono munite di opportuni sensori, *shock sensor*, che permettono di allertare la popolazione se vengono investite da un *debris flow*.
In generale i sistemi d'allerta si possono dividere in due gruppi: avvisi preventivi e avvisi d'evento.
Gli avvisi preventivi si attivano quando sussistono condizioni favorevoli al generarsi di un *debris flow* e si basano su correlazioni empiriche tra le precipitazioni e il verificarsi di un *debris flow*. I sensori adottati sono quelli normalmente utilizzati per il monitoraggio idrometeorologico e sono costituiti da una rete telemetrica di pluviometri e radar meteorologici. La definizione di soglie di piovosità critica per l'innesco di *debris flow* sono inclini ad un'elevata probabilità di falsi allarmi, perché non tutte le precipitazioni che superano queste soglie effetti-

Figura 6: Installazione di una gabbia in località Rosazza, pochi chilometri a nord di Biella.

vamente danno inizio ad un *debris flow*. La difficoltà di individuare con precisione una soglia sta nella complessità del processo di innesco che non può essere esplicitato solo da una relazione con i valori di precipitazione. Come conseguenza questi sistemi di allarme sono adatti per allertare il personale coinvolto nella gestione delle emergenze, ma non per avvertire la popolazione.
Lo scopo degli avvisi d'evento invece è quello di fornire un'allarme quando una colata detritica è già in corso. In questo caso i sensori utilizzati sono spesso gli stessi adottati per il monitoraggio di *debris flow* a scopo di ricerca perchè oltre ad avvisare nel caso si verifichi una colata hanno la possibilità di immagazzinare dati utili per eseguire delle statistiche sull'innesco di *debris flow*; e sono:

ULTRASUONI, RADAR E SENSORI LASER Misurano la fase del flusso, il loro vantaggio è la facilità con cui si riesce ad identificare le soglie d'allarme però devono essere posizionati sul canale, e l'installazione può risultare difficile se il canale è instabile;

GEOFONI E SISMOMETRI Misurano le vibrazioni del terreno causate dal flusso di detriti, l'installazione è facile e sicura, infatti i sensori sono sepolti in luoghi stabili, però può essere molto difficile individuare le soglie d'allarme ed hanno un'alta probabilità di falsi allarmi dovuto ad altre fonti di vibrazione del terreno (il passaggio di treni o camion, cadute di massi, ecc.);

PENDOLI Rilevano il passaggio di detriti attraverso l'oscillazione del pendolo, è un dispositivo semplice e robusto ma l'installazione può risultare difficoltosa in quanto il pendolo deve essere montato sopra il canale.

SENSORI A FILO Rilevano i *debris flow* tramite la rottura del filo, è un dispositivo semplice e robusto ma necessita di essere ripristinato dopo ogni attivazione e ha un'alta probabilità di falsi allarmi

dovuti a circostanze accidentali (passaggio di animali, caduta di alberi, ecc.);

FOTOCELLULE Rilevano il passaggio di *debris flow*, è un tipo di sensore senza contatto quindi non richiede il ripristino dopo ogni attivazione, però non deve venire a contatto con il flusso, quindi necessita di un'attenta installazione.

VIDEOCAMERE Riconoscono il passaggio di *debris flow*, l'installazione è facile e sicura (la telecamera può essere collocata accanto al canale), però la presenza di nebbia o il verificarsi di colate di detriti durante la notte possono complicare l'uso del sistema.

1.2 ANALISI DEL PROBLEMA

Un sistema d'allarme ad evento è stato sviluppato anche da un gruppo di ricerca dell'università di Padova (professor Carlo Gregoretti ed altri ricercatori del dipartimento TeSAF) per il monitoraggio del canale "Pomagagnon" a Fiames, piccola frazione del comune di Cortina d'Ampezzo. I sensori impiegati (geofoni, misuratori di livello ad ultrasuoni, cavi a strappo, telecamere ad infrarossi e pluviometri) trasmettono le misure ad una centralina che è in grado di stabilire se vi è un fenomeno di colata in atto.

Questo lavoro di tesi si prefigge lo scopo di sviluppare un sistema d'allarme ad evento basato sulla realizzazione di una rete di sensori wireless che attraverso misurazioni radio sono in grado di autolocalizzarsi per monitorare la loro posizione e sono dotati di un accelerometro per misurare in modo più immediato e preciso gli eventuali spostamenti. Per un'introduzione sulle reti di sensori si faccia riferimento a [5].

I vantaggi di questo sistema sono molteplici. In primo luogo la facilità di installazione sul campo infatti la rete è composta di nodi fissi (o nodi ancòra) che vanno posizionati, insieme al PC che effettua i calcoli necessari per la localizzazione, sulle sponde del canale di un possibile *debris flow*, quindi in luogo sicuro; e da nodi mobili che vanno fissati a detriti di grandi dimensioni che potrebbero essere soggetti a spostamenti dovuti ad una colata. La scelta di fissare i nodi mobili ai detriti di dimensione maggiore deriva dalla composizione di un *debris flow* che, come precedentemente detto, ha un gradiente inverso di distribuzione granulometrica quindi i detriti di dimensione maggiore tendono ad essere mantenuti in superficie.

Un ulteriore vantaggio è la capacità di operare in qualsiasi condizione atmosferica, quindi anche con limitata visibilità dovuta a nebbie e durante la notte.

Inoltre essendo la rete composta da più sensori risulta facile individuare una soglia d'allarme, ovvero quando più sensori vicini si muovono in modo simile, ed è robusta ai falsi allarmi, infatti se per cir-

costanze accidentali (passaggio di animali, rotolamento di un singolo detrito, ecc.) un sensore dovesse subire degli spostamenti il sistema sarebbe in grado di identificare l'accidentalità dell'accaduto.

Progetti di reti di sensori wireless sono già stati sviluppati e studiati anche per la localizzazione [14] [9] [2], ma questo si differenzia in quanto considera un numero limitato si sensori fissi ed un elevato numero di sensori mobili

1.3 SCHEMA DELLA TESI

La tesi è così suddivisa:
Il **secondo capitolo** presenta la soluzione del sistema, dettagliando gli aspetti teorici dell'algoritmo di localizzazione e la sua implementazione.
Il **terzo capitolo** riporta una analisi sperimentale volta ad individuare le caratteristiche minime che i dispositivi scelti per la realizzazione devono avere.
Il **quarto capitolo** illustra le caratteristiche hardware e software del dispositivo selezionato, il software sviluppato per il suo uso e i risultati sperimentali ottenuti.
L'**ultimo capitolo** conclude la tesi presentando alcune considerazioni e possibili estensioni future.

2

SOLUZIONE AL PROBLEMA DELLA LOCALIZZAZIONE

Come introdotto nel capitolo precedente lo scopo di questo lavoro di tesi è quello di sviluppare un sistema di monitoraggio per colate detritiche attraverso una rete di sensori wireless posizionati nel possibile bacino di deflusso che sono in grado di autolocalizzarsi.
Condizione necessaria per l'utilizzo di questo approccio è la possibilità di avere un algoritmo di localizzazione robusto in grado di determinare le posizioni dei nodi a partire dalle misure di distanza tra i nodi effettuate dai nodi stessi.
Poiché la rete di sensori si pone come obbiettivo la rilevazione di spostamenti del terreno, ogni dispositivo deve inoltre essere dotato di un accelerometro.

In questo capitolo verranno presentati i dettagli dell'algoritmo di trilaterazione che è stato proposto e studiato in questo lavoro di tesi. Esso è in grado di utilizzare le distanze relative tra i sensori per stimare le posizioni dei nodi rispetto ad alcuni sensori fissi (nodi ancòra).
L'algoritmo di trilaterazione proposto è basato su un algoritmo di minimizzazione ai minimi quadrati, infatti lo scopo è quello di stimare la posizione dei nodi minimizzando l'errore tra le distanze misurate e le distanze calcolate a partire dalla stima della posizione.

2.1 ALGORITMO DI TRILATERAZIONE

Si consideri un generico grafo come quello riportato in Figura 7, composto da nodi e archi $\mathcal{G} = \left(V, \xi\right)$.

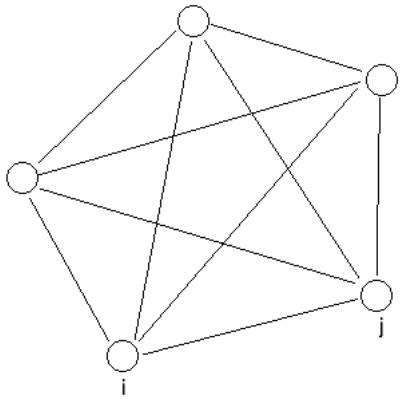


Figura 7: Generico grafo rappresentante una distrubuzione dei nodi.

Dati due nodi i e j si consideri la stima della loro distanza come la somma tra il valore reale e un errore, come riportato il eq. 1.

$$\tilde{d_{ij}} = d_{ij} + n_{ij} \tag{1}$$

Si consideri la funzione

$$f_{ij} = \tilde{d_{ij}}^2 - d_{ij}^2$$

nella quale d_{ij}^2 è definito come:

$$d_{ij}^2 = (x_i - x_j)^2 + (y_i - y_j)^2$$

Lo scopo è quello di minimizzare una funzione costo quadratica $F(x)$ ottenuta sommando i quadrati di f_{ij} per ogni i e per ogni j come riportato in eq. 2.

$$F(x) = \sum_{(i,j)\in\xi} \left(\tilde{d_{ij}}^2 - \left((x_i - x_j)^2 + (y_i - y_j)^2\right)\right)^2 \tag{2}$$

Si vuole quindi trovare la stima $\mathbf{z}^*$ del vettore di stato composto dalle coordinate x, y dei nodi non ancòra, descritto in eq. 3, tale che $\mathbf{F}'(\mathbf{z}^*) = 0$.

$$\mathbf{z} = \begin{pmatrix} \begin{pmatrix} x_0 & y_0 \end{pmatrix} \\ \begin{pmatrix} x_1 & y_1 \end{pmatrix} \\ \vdots \\ \begin{pmatrix} x_n & y_n \end{pmatrix} \end{pmatrix} \tag{3}$$

Per fare ciò si propone di utilizzare il metodo di Newton.

2.1.1 *Metodo di Newton*

Il metodo di Newton, come tutti i metodi per la soluzione di sistemi non lineari, è iterativo; partendo da una condizione iniziale del vettore di stato $\mathbf{z_0}$ produce una serie di vettori $\mathbf{z_1}, \mathbf{z_2}, \ldots$, che (fiduciosamente) convergono a $\mathbf{z}^*$, un minimo locale della funzione data.
Essendo il sistema da risolvere non lineare è necessario eseguire una linearizzazione, quindi si considera lo sviluppo di Taylor della funzione vettoriale $\mathbf{F}'(\mathbf{z})$, ottenendo:

$$\mathbf{F}'(\mathbf{z} + \mathbf{h}) \simeq \mathbf{F}'(\mathbf{z}) + \mathbf{F}''(\mathbf{z})\mathbf{h}$$

dove $\mathbf{h}$ è il vettore da sommare a $\mathbf{z}$ per ottenere il passo successivo dell'iterazione, e si calcola risolvendo il sistema lineare di eq. 4.

$$\mathbf{H}\mathbf{h} = -\mathbf{F}'(\mathbf{z}) \quad \text{con} \quad \mathbf{H} = \mathbf{F}''(\mathbf{z}) \tag{4}$$

Il processo iterativo viene arrestato quando:

$$\|\mathbf{z}(k+1) - \mathbf{z}(k)\| \leqslant \epsilon$$

dove k è il numero dell'iterazione.
Per maggiori informazioni teoriche sugli algoritmi di minimizzazione si faccia riferimetno a [7].

Nel caso in esame si vuole trovare delle coppie di soluzioni $\mathbf{z}_i = (x_i, y_i)$, dalle quali dipende la funzione da minimizzare 2; si distinguono quindi le derivare parziali rispetto alle componenti x e le componenti y.
Il sistema non lineare da risolvere ($\mathbf{F}'(\mathbf{z})$) è quindi composto dalle equazioni riportati in 5 calcolate per ogni nodo appartenente all'insieme dei nodi N, dove N_i è l'insieme dei nodi "vicini" a nodo i.

$$\frac{\partial F}{\partial x_i} = \sum_{j \in N_i} -4 \left[\tilde{d_{ij}}^2 - (x_i - x_j)^2 - (y_i - y_j)^2 \right] (x_i - x_j) \tag{5a}$$

$$\frac{\partial F}{\partial y_i} = \sum_{j \in N_i} -4 \left[\tilde{d_{ij}}^2 - (x_i - x_j)^2 - (y_i - y_j)^2 \right] (y_i - y_j) \tag{5b}$$

Per il calcolo del vettore $\mathbf{h}$ serve conoscere analiticamente la forma di $\mathbf{H}$. Questa è composta da tutte le derivarte seconde di F.
Quindi derivando ulteriormente le equazioni 5 rispetto a tutte le possibili componenti del vettore di stato si ottiene quanto riportato in 6.

$$\frac{\partial^2 F}{\partial x_i^2} = \sum_{j \in N_i} -4 \left[\left(\tilde{d_{ij}}^2 - (x_i - x_j)^2 - (y_i - y_j)^2 \right) - 2(x_i - x_j)^2 \right] \tag{6a}$$

$$\frac{\partial^2 F}{\partial x_i \partial x_j} = 4 \left[\left(\tilde{d_{ij}}^2 - (x_i - x_j)^2 - (y_i - y_j)^2 \right) - 2(x_i - x_j)^2 \right] \tag{6b}$$

$$\frac{\partial^2 F}{\partial x_i \partial y_i} = \sum_{j \in N_i} 8(x_i - x_j)(y_i - y_j) \tag{6c}$$

$$\frac{\partial^2 F}{\partial x_i \partial y_j} = -8(x_i - x_j)(y_i - y_j) \tag{6d}$$

$$\frac{\partial^2 F}{\partial y_i \partial x_i} = \sum_{j \in N_i} 8(x_i - x_j)(y_i - y_j) \tag{6e}$$

$$\frac{\partial^2 F}{\partial y_i \partial x_j} = -8(x_i - x_j)(y_i - y_j) \tag{6f}$$

$$\frac{\partial^2 F}{\partial y_i^2} = \sum_{j \in N_i} -4 \left[\left(\tilde{d_{ij}}^2 - (x_i - x_j)^2 - (y_i - y_j)^2 \right) - 2(y_i - y_j)^2 \right] \tag{6g}$$

$$\frac{\partial^2 F}{\partial y_i \partial y_j} = 4 \left[\left(\tilde{d_{ij}}^2 - (x_i - x_j)^2 - (y_i - y_j)^2 \right) - 2(y_i - y_j)^2 \right] \tag{6h}$$

2.2 IMPLEMENTAZIONE

La realizzazione dell'algoritmo di trilaterazione introdotto viene sviluppata in C++.

Per mantenere il progetto strutturato ad oggetti, condizione necessaria per l'utilizzo delle *boost thread*, è stata creata una classe che descrive le condizioni del sistema, quindi: il numero di ancore, le posizioni stimate dei nodi e le distanze misurate tra di essi. I metodi di questa classe, oltre ai costruttori e distruttori, sono due. Il primo, `void operator ()()`, serve per il mutithreading ed è il "main" della thread che gestisce la trilaterazione, mentre l'algoritmo è implementato dal metodo `void loc()`.
Questa classe permette l'utilizzo del programma con un numero di nodi variabile fino ad un massimo di `MAX_MOTE`, costante e definito nel file *parametri.h*. Quindi è possibile aggiungere un nodo mentre il programma è in esecuzione senza dover ricompilare il codice e se un nodo andasse perso, perchè danneggiato o con batterie scariche o fuori dalla portata della comunicazione radio, il processo non verrebbe interrotto ma continuerebbe il calcolo della stima di posizione con un nodo in meno.

Per lo sviluppato il sistema utilizza i contenitori *vector* delle librerie STL e, come già anticipato, le *boost thread*.
Le distanze misurate sono memorizzate dalla thread che si occupa della comunicazione e gestione dei sensori (thread server) in un `vector` bidimensionale condiviso con la thread che si occupa della stima delle posizioni (thread trilaterazione). In questa struttura ogni posizione riporta la distanza misurata tra i sensori, quindi in posizione ij si trova la distanza tra il nodo i e il nodo j misurata dal nodo i sui livelli di potenza dei pacchetti ricevuti da j. Sulla diagonale principale e nelle posizioni ij con nodo i o nodo j mancante o per il quale non ci siano misurazioni relative all'altro nodo il valore viene impostato a 0.
Le due thread non possono accedere contemporaneamente alla struttura condivisa per cui l'accesso è gestito da *mutex* e *condition variable*.
La struttura condivisa può essere sovrascritta in qualsiasi momento dalla thread server, anche se la thread trilaterazione non ha ancora letto i dati, infatti non interessa avere una stima di posizione per ogni misura di distanze effettuata, quanto la stima della posizione per l'ultima misurazione. Di conseguenza la thread server può scrivere sulla stuttura, dopo aver controllato se la thread trilaterazione non sta leggendo il contenuto, ogni volta che ci sono dei nuovi dati di distanze misurate.
La thread trilaterazione, invece, legge i dati solo quando sono nuovi quindi oltre a controllare che la thread server non stia scrivendo sulla struttura controlla che i dati non siano già stati letti. Per fare ciò è utilizzata una variabile booleana condivisa che viene posta a *true* dalla thread server ogni volta che questa inserisce dei dati nuovi e la thread trilaterazione la pone a *false* ogni volta che legge dei dati. Questa variabile condivisa è protetta da un *mutex* in entrambe le thread ed è utilizzata con una *condition variable* nella thread trilaterazione.

Per quanto riguarda la thread trilaterazione la gestione della memoria condivisa con tutti i controlli è implementata dal metodo `void operator()()`. Di seguito viene riportato un estratto di questo metodo nel quale si vede l'impiego del *mutex* che gestisce l'accesso concorrenziale alle variabili condivise e la *condition variable* che gestisce la sincronizzazione delle lettura della struttura condivisa contenente le distanze misurate.

```
24 mutexD.lock();                  //Blocca ad altre thread l'utilizzo
                                     //delle variabili condivise
26 while (!(new_data))             //Verifica se ci sono nuovi dati
     condGo.wait(mutexD);          //Viene liberato mutexD e la thread si
28                                   //addormenta fino a quando quanche
                                     //thread non fa signal su mutexD
30 distanze_=distanze_condivise;   //Vengono copiati i valori di distanza
                                     //nella variabile membro
32 new_data=false;                 //Segnala che i dati sono stati letti
   mutexD.unlock();                //Libera mutexD
```

Si ricorda che il comando `wait` invocato su una *condition variable* sblocca il *mutex* associato, nel nostro caso `mutexD`, ed addormenta la thread fino a quando un altra thread non invoca il comando `signal` sulla stessa *condition variable*. Quando ciò avviene la thread si attiva e blocca il *mutex* associato alla *condition variable*; se ci sono dati nuovi da leggere, condizione indicata da `new_data`, la thread procede mantenendo il *mutex* bloccato fino a quando non viene liberato dal comando `unluck()`, in caso contrario viene chiamato nuovamente il comando `wait`.

Il metodo `void loc()` implementa l'algoritmo di localizzazione; si occupa quindi di iterare i calcoli riportati in 4 fino ad ottenere una stima delle posizioni. Per comodità nello sviluppo del codice il vettore delle posizioni dei nodi da stimare non è nella forma riportata in 3, ma è diviso in due vettori separati; uno per le coordinate x e l'altro per le coordinate y. Così facendo le coordinate di un generico nodo i sono memorizzate nella posizione i dei rispettivi vettori; quindi, se per esempio consideriamo il nodo 3 le sue coordinate $\begin{pmatrix} x & y \end{pmatrix}$ sono memorizzate rispettivamente in $x(3)$ e $y(3)$.
Tuttavia il vettore di stato z utilizzato per il calcolo delle variazioni delle coordinate per ogni iterazione h, riportato in 4, deve essere unico e contenere entrambe le coordinate di tutti i nodi. Questo vettore

è ottenuto accodando i due vettori delle coordinate x e y, ottenendo un vettore nella forma riportata in 7.

$$\mathbf{z} = \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_{nNodi} \\ y_1 \\ y_2 \\ \vdots \\ y_{nNodi} \end{pmatrix} \tag{7}$$

Questa scelta progettuale è stata fatta per semplificare il calcolo del vettore $F'(z)$ e della matrice $F''(z)$. Infatti se la derivata rispetto ad x di un nodo è in posizione i la derivata rispetto ad y dello stesso nodo è in posizione $i + nNodi$, dove $nNodi$ rappresenta il numero di nodi effettivamente rilevati.

Come prima cosa, per poter gestire un numero variabile di nodi, il metodo `void loc()` analizza la matrice delle distanze per capire quali mote non ancòre ci sono e crea un vettore *didascalia* il quale lega l'ID del mote alla sua posizione del vettore di stato.
Questa routine verifica la presenza di un nodo i controllando se nella matrice delle distanze nelle posizioni realtive ad i, quindi in tutte le posizioni ij e ji con j compreso tra 0 e il numero massimo di nodi, ci sono delle misure di distanza. Infatti se in nodo non dovesse essere presente tutte le distanze che lo riguardano sarebbero a 0.
Per ogni nodo presente la routine aggiunge al vettore didascalia una cella contenente l'ID del nodo stesso, ottenendo così un vettore che lega l'ID dei nodi con la posizione delle loro coordinate all'interno del vettore di stato.
Se, per esempio, il sistema supporta 10 nodi, da 0 a 9, di cui i primi tre ancòre e i nodi 5 e 8 assenti, il vettore didascalia e il vettore di stato sono quelli riportati in 8.

Dopo aver creato il vettore didascalia il metodo `void loc()` comincia le iterazioni. Per ognuna di queste calcola il vettore F' e la matrice F'' applicando rispettivamente le equazioni 5 e 6 e utilizzando i valori memorizzati nella matrice delle distanze. Per aver la certezza di utilizzare tutti i valori di distanza misurati, anche quelli misurati dalle ancòre, nelle equazioni viene utilizzata la media delle misure nei due versi; cioè, $\tilde{d_{ij}}$ viene sostituita con la media di $\tilde{d_{ij}}$ e $\tilde{d_{ji}}$. Questo viene fatto perchè che nel vettore di stato non sono presenti le coordinate dei nodi ancòra, essendo queste note e non incognite, di conseguenza non vengono considerare le righe della matrice delle distanze relative a questi nodi; senza questo accorgimento non verrebbero quindi considerati i valori di distanza in esse contenuti.

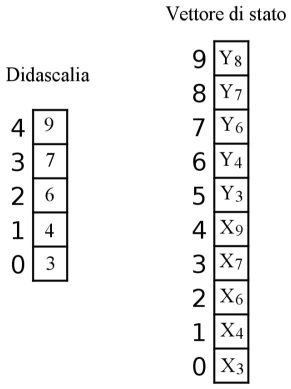


Figura 8: Esempio di vettore didascalia e vettore di stato associato

Una volta ottenuta la matrice F'' ne viene calcolata l'inversa e viene applicata l'equazione 4 per il calcolo di h; questa è un vettore nella forma 7. I valori contenuti in h vengono sommati alle coordinate dei nodi memorizzate nelle variabili membro della classe.

Le iterazioni continuano fino a quando la somma dei quadrati dell'incremento diventa minore di un valore ε impostato in fase di compilazione o fino ad un numero massimo di iterazioni. La prima condizione si formalizza con la seguente disuguaglianza:

$$\sum_{i=0}^{2*nNodi} h_i^2 < \varepsilon$$

Il metodo `void loc()` non restituisce valori ma opera direttamente sui membri della classe, di conseguenza quando conclude la stima delle coordinate dei nodi queste sono già memorizzate nelle variabili membro.

3

ANALISI IN SIMULAZIONE

Per valutare il comportamento dell'algoritmo di trilaterazione in presenza di errori nelle misure sensoriali è stata compiuto uno studio in simulazione. La prima serie di simulazioni ha l'obbiettivo di valutare l'algoritmo in condizioni statiche. I nodi vengono posti in posizione fissa e si valuta la precisione dell'algoritmo di trilaterazione in presenza di errori nelle condizioni iniziali e nella valutazione delle distanze. La seconda serie di simulazioni, invece, valuta l'algoritmo in condizioni dinamiche. Per fare ciò vengono considerate alcune configurazioni dei nodi e, per ognuna di queste, viene variata in modo lineare e determinato la posizione di uno dei nodi.

Nella prima serie di simulazioni si sono considerate tre configurazioni di 11 nodi con diverse posizioni per i nodi o con diverso numero di ancòre:

CONFIGURAZIONE 1: 3 ancòre (nodi 0, 1 e 2) e mote posizionati in modo ordinato (Figura 9);

CONFIGURAZIONE 2: 4 ancòre (nodi 0, 1, 2 e 10) e mote posizionati in modo ordinato (Figura 9);

CONFIGURAZIONE 3: 3 ancòre (nodi 0, 1 e 2) e mote posizionati in modo casuale all'interno dello spazio considerato (Figura 10).

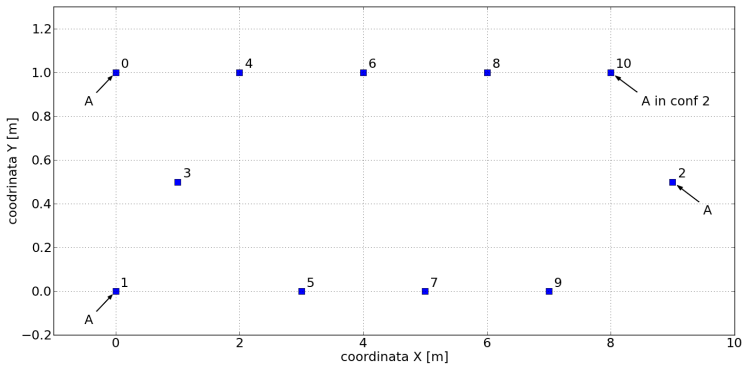


Figura 9: Posizione dei nodi nelle prime due configurazioni.

Si considerino le disposizioni iniziali dei nodi come mostrato in Figura 9 e 10 nelle quale l'asse delle ascisse rappresenta la coordinata x e l'asse delle ordinate rappresenta la coordinata y. Nei grafici riportati vengono inoltre indicati con la lettera "A" i nodi ancòra.
La scelta di validare l'algoritmo con i nodi così disposti deriva da

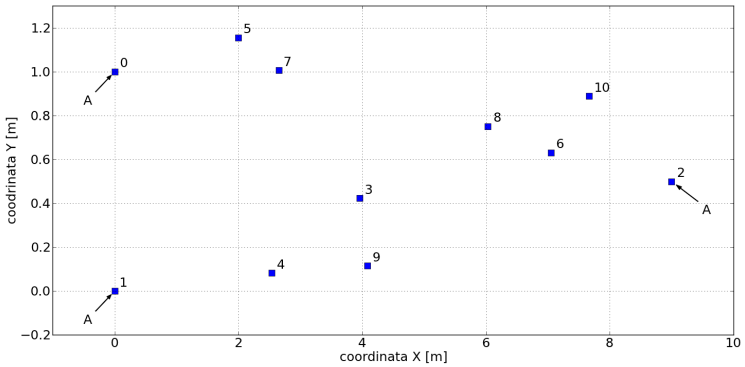


Figura 10: Posizione dei nodi per la configurazione 3.

considerazioni sul loro probabile posizionamento in situazione reale. Infatti, ipotizzando il canale verticale con una larghezza di 8m (larghezze tipiche vanno da 6m a 10m) i mote ancòra risulterebbero posizionati sulle sponde del canale stesso, mentre i mote mobili sarebbero disposti nel canale lungo il fronte di avanzamento del *debris flow*.

Le coordinate dei nodi vengono riportare in Tabella 1 per le configurazioni 1 e 2 e in Tabella 2 per la configurazione 3.
Le condizioni iniziali per l'algoritmo di trilaterazione utilizzano le coordinate esatte a cui viene aggiunto un errore di tipo gaussiano a media nulla e deviazione standard γ. Stimando, senza l'ausilio di strumenti di misura precisi, le coordinate dei nodi in una possibile situazione reale è plausibile considerare un errore di stima con $\gamma = 40\text{cm}$.
Per quanto riguarda le distanze tra i mote sono state prese le distanze esatte, calcolate a partire dalle coordinate dei nodi riportate in Tabella 1 e 2, a cui viene aggiunto un errore, sempre di tipo gaussiano, con media nulla e deviazione standard δ, che viene variato nelle diverse prove per valutare l'errore massimo sostenibile dall'algoritmo di trilaterazione.

Per ogni configurazione sono stati fatti 5 test modificando il valore di δ, considerando i seguenti casi: 5cm, 10cm, 20cm, 40cm e 80cm. Ogni test consiste nel ripetere per N volte ($N = 100$) l'algoritmo di trilaterazione valutando, per ogni ripetizione, la media dell'errore tra la posizione esatta e la posizione stimata tra i nodi. Formalmente, ad ogni ripetizione viene eseguito il calcolo seguente:

$$\bar{\varepsilon} = \frac{\sum_{i=1}^{N\_MOTE} \overline{Ms_i - M_i}}{N\_MOTE}$$

dove, $\bar{\varepsilon}$ rappresenta l'errore medio, N_MOTE rappresenta il numero dei nodi presenti (11 nel nostro caso) e $\overline{Ms_i - M_i}$ rappresenta la distanza geometrica tra la posizione esatta (M_i) e quella stimata (Ms_i) per il nodo i.

Tabella 1: Coordinate dei mote nelle configurazioni 1 e 2.

N° mote	x	y
0 (ancòra)	0	1
1 (ancòra)	0	0
2 (ancòra)	9	0.5
3	1	0.5
4	2	1
5	3	0
6	4	1
7	5	0
8	6	1
9	7	0
10 (ancòra in conf. 2)	8	1

Tabella 2: Coordinate dei mote nella configurazione 3.

N° mote	x	y
0 (ancòra)	0	1
1 (ancòra)	0	0
2 (ancòra)	9	0.5
3	3.96	0.42
4	2.53	0.08
5	1.99	1.15
6	7.0	0.63
7	2.65	1.01
8	6.03	0.75
9	4.08	0.11
10	7.67	0.89

Dopo aver eseguito le N ripetizioni si calcola la media e la deviazione standard tra tutti gli N errori medi ottenuti in ogni test.

Come precedentemente introdotto, la seconda serie di simulazioni ha l'obbiettivo di valutare l'algoritmo in condizioni dinamiche.
Per questa serie di simulazioni si sono posizionati i mote come in Figura 10 e si è variata la posizione del mote 6 aumentando la sua ordinata di 50cm in più passi. Una prova è stata fatta modificando la posizione del mote in 10 passi da 5cm, e nell'altra prova in 5 passi da 10cm l'uno. Per ognuno di questi passi è stato utilizzato l'algoritmo di trilaterazione.
Al primo passo di ogni prova vengono considerati i mote nella disposizione in Figura 9. Le condizioni iniziali, come nella prima serie di simulazioni, sono le coordinate esatte dei nodi alle quali viene aggiunto un errore di tipo gaussiano a media nulla e deviazione standard γ, con $\gamma = 40\text{cm}$.
Per i passi successivi vengono utilizzate come condizioni iniziali dell'algoritmo di trilaterazione le posizioni stimate al passo precedente.
Le distanze usate per la localizzazione sono le distanze esatte di ogni passo alterate di un errore gaussiano a media nulla e deviazione standard δ. Ancora una volta vengono fatte diverse analisi con δ che assume valori pari a 5 e 10 cm, unici valori significativi come mostrato dalla serie di simulazioni in situazione statica.
Per ogni prova vengono eseguiti N test ($N = 100$); per ogni test viene calcolato l'errore di stima medio sul numero di passi per il nodo mobile (mote 6) e l'errore di stima della posizione del nodo mobile per tre particolai passi. Quest'ultimo calcolo viene effettuato per verificare se l'eventuale errore si accumula lungo il tragitto. Dopo aver eseguito tutti gli N test viene calcolata la media e la deviazione standard dell'errore medio di stima, e la media e la deviazione standard sulle N prove dell'errore di stima per i tre passi considerati.

3.1 PRIMA SERIE DI SIMULAZIONI: CONFIGURAZIONI STATICHE, ERRORI DI MISURA CRESCENTI

I test che compongono la prima serie di simulazioni considerano la situazione statica ed includono tutte e tre le configurazioni. Per ogni configurazione viene valutata la precisione ottenibile con diversi valori di δ.
Per ogni test vengono riportati i valori di media e deviazione standard (in metri) dell'errore ed un grafico riportante le posizioni dei mote esatte, quelle iniziali afflitte da errore con $\gamma = 40\text{cm}$ e le posizioni stimate.

3.1.1 *Configurazione 1: disposizione ordinata dei mote con 3 ancòre*

3.1.1.1 $\delta = 5cm$

In questo caso viene considerata una deviazione standard (δ) dell'errore di misura delle distanze piccola rispetto all'ordine di grandezza delle distanze stesse.
Come si può notare da Tabella 3 l'errore medio stimanto su N prove è basso.
La Figura 11 riporta la situazione iniziale e la stima finale dell'algoritmo in uno degli N test eseguiti nella prima configurazione. In questo esempio, malgrado all'inizio ci sia una valutazione errata delle posizioni iniziali dei mote, l'algoritmo è in grado di far convergere le misure a valori vicini alle posizioni reali dei mote.

Tabella 3: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 5cm$

Media errore medio	0.10
Deviazione standard	0.03

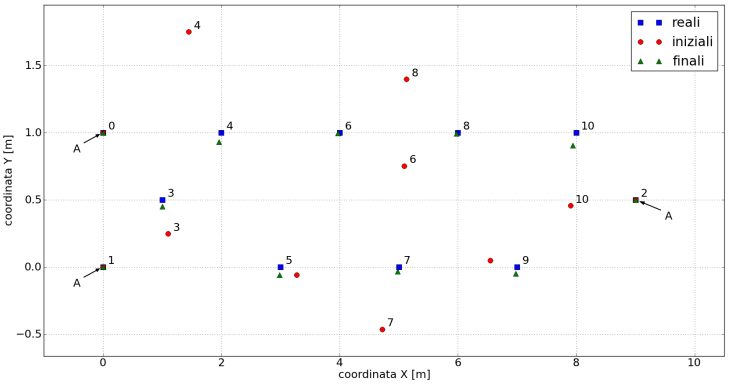


Figura 11: Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 5$.

3.1.1.2 $\delta = 10cm$

Anche con questo valore di δ il risultato ottenuto è stato buono, come si può notare dalla media dell'errore riportata in Tabella 4, ma aumenta decisamente la deviazione standard. Questo porta a situazioni come quella in Figura 12 dove le posizioni finali stimate per alcuni nodi sono peggiori di quelle considerate come valore iniziale, nodi 9 e 10.

Tabella 4: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 10cm$

Media errore medio	0.18
Deviazione standard	0.19

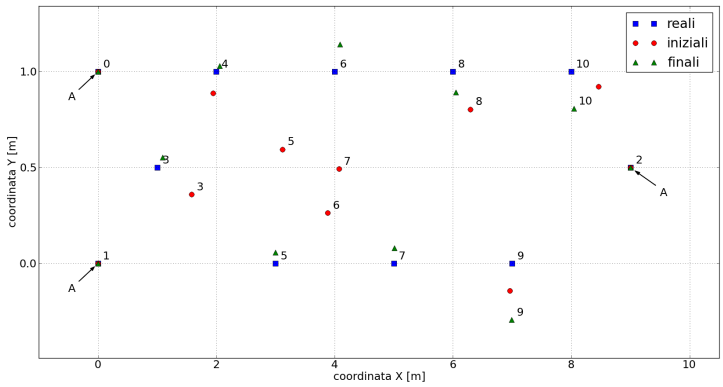


Figura 12: Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 10cm$

3.1.1.3 $\delta = 20cm$

Con questo test si comincia ad avere una media dell'errore significativa, infatti è di $29cm$, quasi un terzo della distanza minima tra i nodi. Tuttavia la Figura 13 risulta essere ingannevole, infatti rapresenta un caso con errore medio di $71cm$. Un errore così elevato è giustificato ricordando che il 99.7% dei valori su cui si calcola la media e la deviazione standard cadono in un intervallo di $\pm 3\sigma$, con σ deviazione standard, attorno alla media, in questo caso in un intervallo $[-0.19m; 0.77m]$. É stato scelto di riportare questo esempio perché rappresenta un problema di stima che si verifica abbastanza frequentemente, soprattutto per valori di δ alti.
Questa stima così errata che porta ad avere i nodi con coordinate y più basse superare i nodi in posizione reale superiore potrebbe essere dovuta alla presenza di un solo nodo ancòra sul lato destro. Difatti avendo due mote fissi sul lato destro la posizione stimata dei nodi mobili più vicini è più vincolata e quindi con meno errori.

3.1.1.4 $\delta = 40cm$ *e* $\delta = 80$

Questi ultimi due test sono stati riportati insieme perchè entrambi restituiscono una media dell'errore alta. Con $\delta = 40cm$ si ottiene una media dell'errore medio di $49cm$ che considerando le distanze minime tra i mote rende l'algoritmo inutilizzabile.

Tabella 5: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 20cm$

Media errore medio	0.29
Deviazione standard	0.16

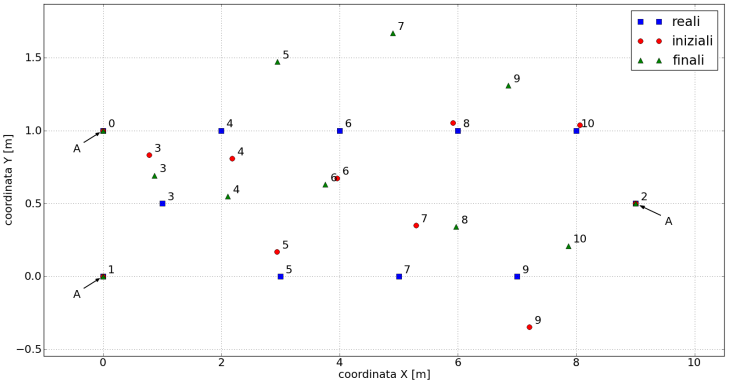


Figura 13: Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 20cm$

Da notare Figura15, che come nel caso riportato per il test con $\delta = 20cm$, rappresenta una stima difettosa a causa, forse, della mancanza di un ulteriore punto fisso.

Tabella 6: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 40cm$

Media errore medio	0.49
Deviazione standard	0.17

3.1.2 *Configurazione 2: disposizione ordinata dei mote con 4 ancòre*

Questa serie di test è stata eseguita per verificare l'ipotesi fatta in 3.1.1.3. Si intende quindi verificare se l'aggiunta di un ancòra sul lato destro della configurazione evita stime in cui le posizioni verticali dei nodi sono ribaltate.

3.1.2.1 $\delta = 5cm$

Come nel caso con sole 3 ancòre questo test ha dato dei risultati soddisfacienti, infatti il valore dell'errore medio stimato su N prove riportato in Tabella 8 è molto basso. Nell'esempio riportato in Figura16 si

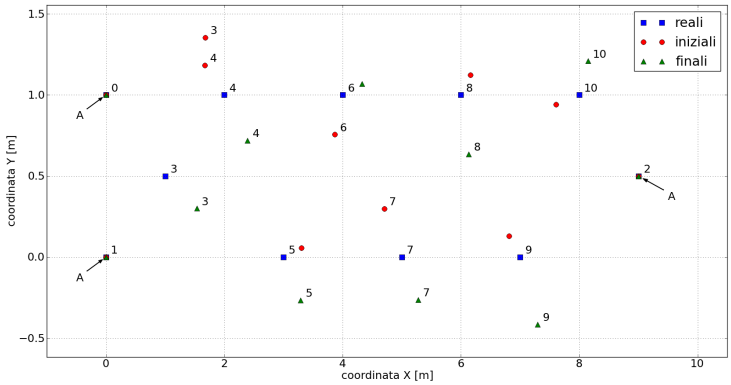


Figura 14: Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 40cm$

Tabella 7: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione ordinata e $\delta = 80cm$

Media errore medio	0.72
Deviazione standard	0.17

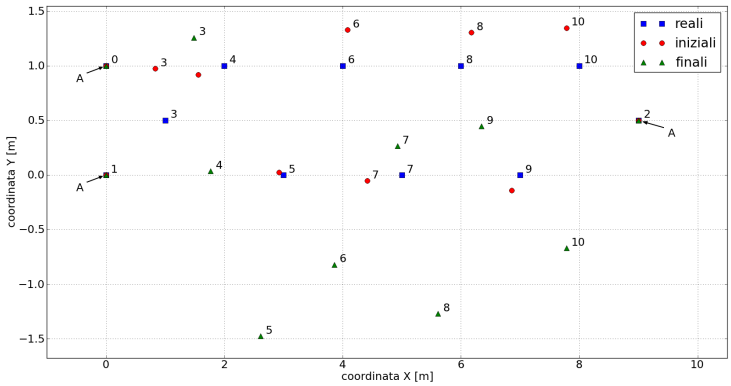


Figura 15: Esempio per uno degli N test con 3 ancòre, disposizione ordinata e $\delta = 80cm$

vede che le posizioni stimate sono andate a convergere sulle posizioni reali.

Tabella 8: Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 5cm$

Media errore medio	0.05
Deviazione standard	0.05

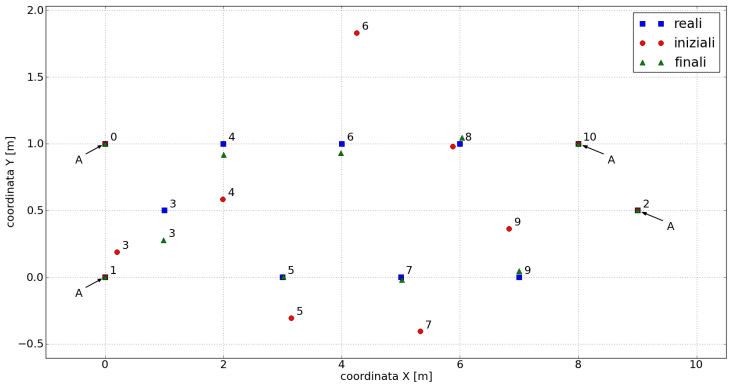


Figura 16: Esempio per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 5cm$

3.1.2.2 $\delta = 10cm$

Sono valide osservazioni analoghe al caso precedente.

Tabella 9: Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 10cm$

Media errore medio	0.09
Deviazione standard	0.03

3.1.2.3 $\delta = 20cm$

Come per il caso con 3 ancòre si nota che con $\delta = 20cm$ l'algoritmo comincia ad avere le prime difficoltà; infatti non si ha più una buona stima, avendo una media d'errore, riportata in Tabella10, abbastanza elevata.

Il test con 4 ancòre e $\delta = 20cm$ è stato eseguito anche utilizzando le condizioni iniziali e le distanze del caso riportato in 3.1.1.3. Questo test è stato eseguito per verificare se con l'aggiunta di un ancòra,

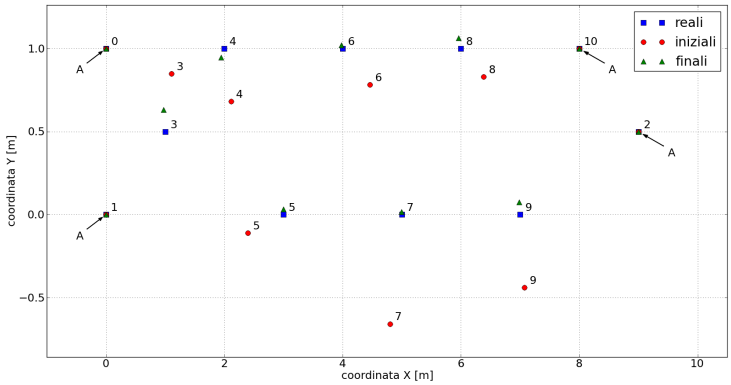


Figura 17: Esempio per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 10cm$

Tabella 10: Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 20cm$

Media errore medio	0.24
Deviazione standard	0.13

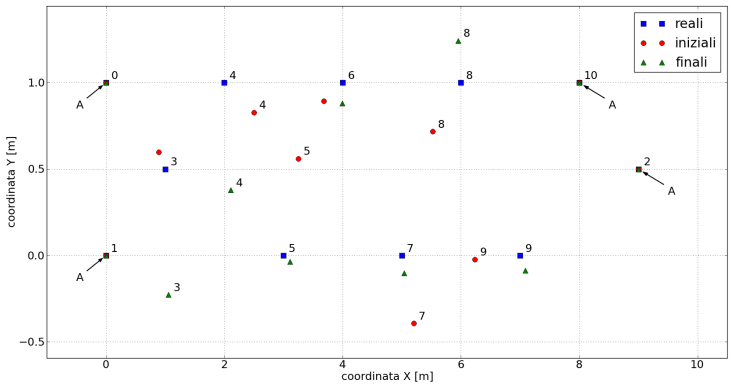


Figura 18: Esempio per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 20cm$

ma con le stesse condizioni iniziali e con gli stessi valori di distanze, l'algoritmo evolve in modo diverso dal caso esempio con 3 ancòre, portando ad una stima che non presenti i mote con coordinate y più basse sopra i mote in posizioni reali superiori. Il grafico ottenuto è riportato in Figura19.

Come si può facilmente notare la situazione non è migliorata, di con-

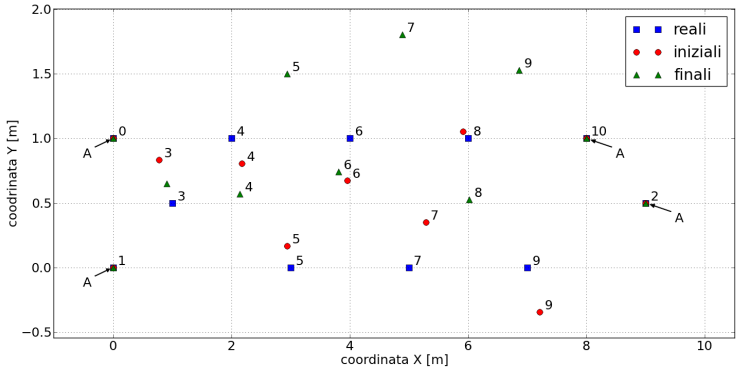


Figura 19: Risultato della prova con condizioni iniziali e distanze del caso esempio di 3.1.1.3 ma con l'aggiunta di un ancòra.

seguenza il problema non è da attribuire alla mancanza di un'ancòra.

3.1.2.4 $\delta = 40cm$ *e* $\delta = 80cm$

Come già ci si aspettava valutando il test precedente, aumentanto ulteriormente il valore dell'errore aggiunto alle distanze l'algoritmo non riesce a convergere ai valori esatti ottenendo una media d'errore, seppur minore al caso con 3 ancòre, molto elevata.

Tabella 11: Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 40cm$

Media errore medio	0.48
Deviazione standard	0.17

Tabella 12: Media e deviazione standard dell'errore medio con 4 ancòre, disposizione ordinata e $\delta = 80cm$

Media errore medio	0.67
Deviazione standard	0.19

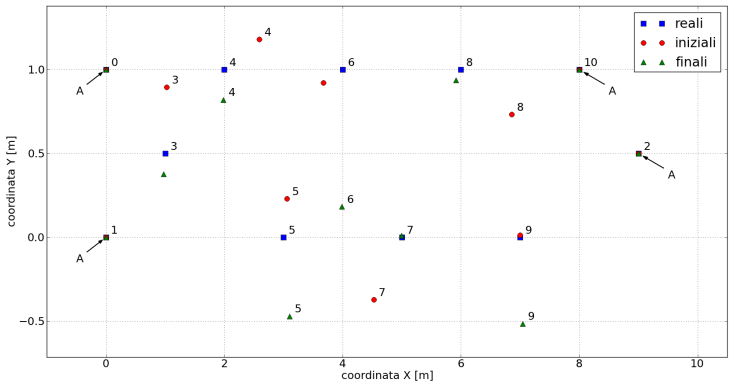


Figura 20: Esempio per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 40cm$

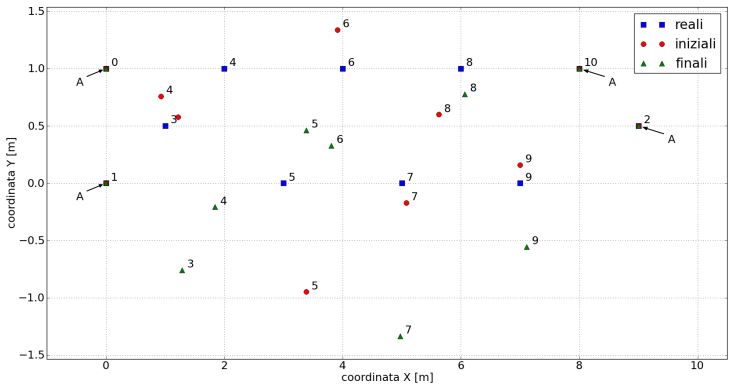


Figura 21: Situazione riassuntiva per uno degli N test con 4 ancòre, disposizione ordinata e $\delta = 80cm$

3.1.3 *Configurazione 3: disposizione casuale dei mote con 3 ancòre*

La configurazione 3 con i mote disposti in modo casuale è stata valutata per verificare la robustezza dell'algoritmo di trilaterazione nel caso in cui i nodi mobili non siano disposti in modo da formare una rete regolare. Questa situazione può verificarsi in più condizioni; i mote possono essere stati posizionati in modo ordinato ma in seguito ad uno smottamento o al passaggio di un animale la loro formazione regolare può essere modificata, oppure, a causa della difficoltà di raggiungere alcuni punti del canale, può rivelarsi impossibile posizionare i mote in modo ordinato e quindi essere costretti a collocarli in modo casuale.

3.1.3.1 $\delta = 5cm$

Seppur con una media d'errore maggiore rispetto al caso con 4 ancòre, anche con la disposizione random questo test ha prodotto degli ottimi risultati, paragonabili al caso con 3 ancòre e disposizione ordinata.

Tabella 13: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 5cm$

Media errore medio	0.11
Deviazione standard	0.08

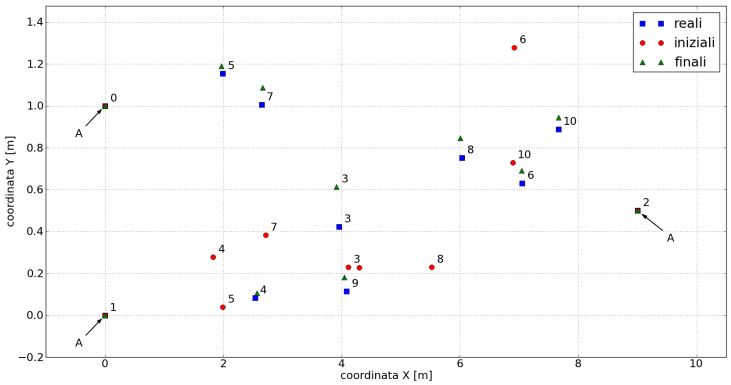


Figura 22: Esempio per uno degli N test con 3 ancòre, disposizione random e $\delta = 5cm$

3.1.3.2 $\delta = 10cm$

Con $\delta = 10cm$ non ci sono notevoli peggioramenti rispetto al caso con $\delta = 5cm$. Per quanto riguarda al confronto con le altre configurazioni le considerazioni sono anologhe al caso precedente.

Tabella 14: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 10cm$

Media errore medio	0.18
Deviazione standard	0.09

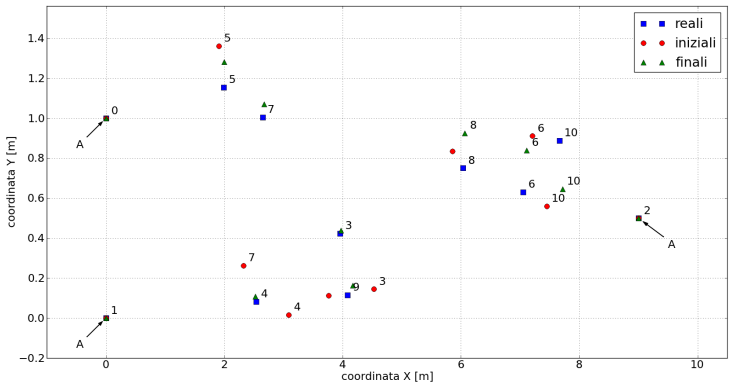


Figura 23: Esempio per uno degli N test con 3 ancòre, disposizione random e $\delta = 10cm$

3.1.3.3 $\delta = 20cm$

Anche per questa configurazione si può dire che $\delta = 20cm$ sia il limite minimo per non considerare più molto attendibile la stima fornita dall'algoritmo di triangolazione.
I risultati forniti per gli N test portano ad avere una media dell'errore medio e una deviazione standard molto vicine a quelle delle due configurazioni viste precedentemente.

Tabella 15: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 20cm$

Media errore medio	0.30
Deviazione standard	0.13

3.1.3.4 $\delta = 40cm$ *e* $\delta = 80cm$

Per questi due test non ci sono particolari osservazioni da fare. Infatti confrontando i valori riportati in Tabella16 e in Tabella 17 con i valori ottenuti dalle corrispettive prove delle due configurazioni precedenti si conclude che all'aumentare dell'errore di distanza introdotto la disposizione dei nodi non ha nessuna influenza sul risultato.

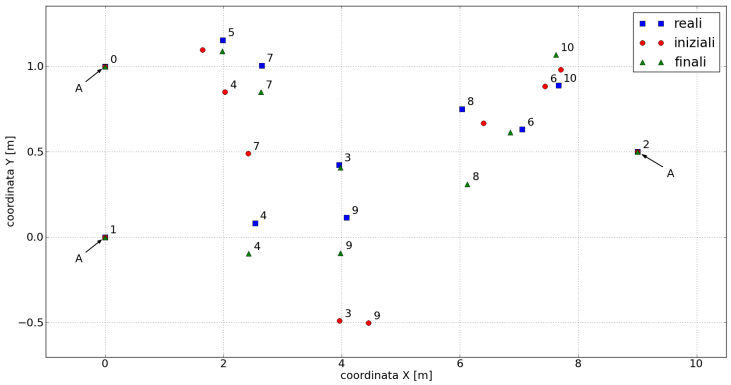


Figura 24: Situazione riassuntiva per uno degli N test con 3 ancòre, disposizione random e $\delta = 20cm$

Tabella 16: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 40cm$

Media errore medio	0.46
Deviazione standard	0.14

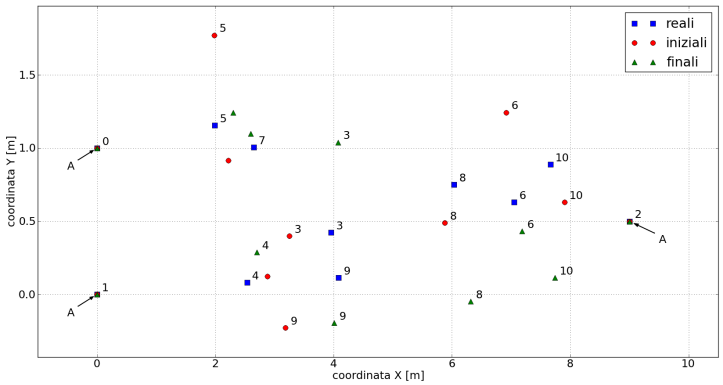


Figura 25: Situazione riassuntiva per uno degli N test con 3 ancòre, disposizione random e $\delta = 40cm$

Tabella 17: Media e deviazione standard dell'errore medio con 3 ancòre, disposizione random e $\delta = 80cm$

Media errore medio	0.73
Deviazione standard	0.20

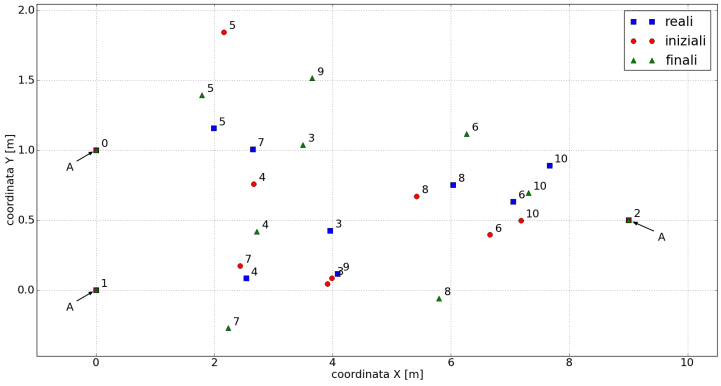


Figura 26: Situazione riassuntiva per uno degli N test con 3 ancòre, disposizione random e $\delta = 80cm$

3.1.4 *Conclusioni sulla valutazione sperimentale nel caso statico*

Tabella 18: Tabella riassuntiva con i valori di errore medio e deviazione standard per le tre configurazioni.

	$\delta = 5cm$		$\delta = 10cm$		$\delta = 20cm$		$\delta = 40cm$		$\delta = 80cm$	
	$\bar{\varepsilon}$	σ	$\bar{\varepsilon}$	σ	$\bar{\varepsilon}$	σ	$\bar{\varepsilon}$	σ	$\bar{\varepsilon}$	σ
Conf 1	0.10	0.03	0.18	0.19	0.29	0.16	0.49	0.17	0.72	0.17
Conf 2	0.05	0.05	0.09	0.03	0.24	0.13	0.48	0.17	0.67	0.19
Conf 3	0.11	0.08	0.18	0.09	0.30	0.13	0.46	0.14	0.73	0.20

Il test fatto in 3.1.2.3 per verificare l'ipotesi che l'aggiunta di un nodo ancòra possa evitare stime errate, in cui i nodi con coordinate y più basse superano i nodi in posizione reale superiore (caso riportato in 3.1.1.3), ha dimostrato che il problema non è il numero di ancòre; infatti il test, eseguito con gli stessi dati ma mantenedo fermo un nodo in più (in nodo 10), ha restituito sostanzialmente la stessa stima finale. Le posizioni stimate dei nodi risultano essere anche nel caso con 4 ancòre ribaltate, nonostante nei due test l'algoritmo di trilaterazione sia evoluto in modo diverso; infatti nel caso con 3 ancòre l'algoritmo è giunto a convergenza con 16 iterazioni con un errore di stima medio tra i nodi di 71cm, mentre nel caso con 4 ancòre l'algoritmo è giunto a convergenza con più iterazioni, 24, ma con un errore di stima medio tra i nodi inferiore, 63cm.

La Tabella 18 riporta i valori di errore medio ($\bar{\varepsilon}$) e deviazione standard (σ) per tutte le configurazioni, permettendo una facilità di confronto. Analizzando i dati contenuti in questa tabella si possono trarre alcune conclusioni.

Innanzitutto, come già osservato, aumentando troppo l'errore introdotto nelle misure di distanza, quindi con δ uguale a 40 e 80 centimetri, non ha più rilevanza nè la disposizione dei nodi, nè il numero di

ancòre.
Per valori dell'errore di misura della distanza piccoli, quindi con δ uguale a 5 e 10 centimetri, l'algoritmo converge ad una buona soluzione in tutte le configurazioni. Si può però notare che a parità di nodi ancòra la disposizione dei nodi mobili non influenza il risultato, infatti l'errore medio nelle due configurazioni con 3 ancòre (configurazioni 1 e 3) sono molto simili. Mentre si può osservare che a parità di disposizione dei nodi (configurazioni 1 e 2) l'algoritmo si comporta meglio all'aumentare del numero di ancòre, essendo il sistema più vincolato.
Per quanto riguarda il caso con l'errore di misura delle distanze pari a $20cm$ si può confermare l'osservazione nella quale si considera tale caso come limite per la buona riuscita della triangolazione. Si può notare che l'errore medio aumenta notevolmente rispetto ai casi con δ inferiore ed inoltre si comincia a perdere l'influenza positiva dell'aggiunta di un'ancòra. Infatti confrontando gli errori medi delle tre configurazioni si osserva che mentre nei casi con δ uguale a 5 e 10 centimetri l'errore della configurazione 2 è la metà rispetto all'errore medio delle configurazioni 1 e 3 con δ uguale a 20 centimetri l'errore della configurazione 2, seppur minore, non è molto diverso dalle altre due configurazioni.

3.2 SECONDA SERIE DI SIMULAZIONI: CONFIGURAZIONI DINAMICHE

I risultati ottenuti nella sezione precedente portano a limitare le analisi successive a due soli valori di delta: $\delta = 5cm$ e $\delta = 10cm$.
Per l'analisi di questa seconda serie di simulazioni, eseguite per verificare la robustezza dell'algoritmo all'inseguimento di un eventuale spostamento dei nodi, vengono riportati i valori di media e deviazione standard dell'errore di stima media e dell'errore di stima per i tre passi presi in esame. Per ogni prova vengono inoltre riportati due grafici; uno rappresentante la variazione di posizione del solo nodo mobile (mote 6), ed uno raffigurante la situazione globale con le variazioni di stima di tutti i nodi presenti.

3.2.1 *Configurazione 1: Spostamento del nodo 6 con 10 passi da 5cm*

Questa prova prevede lo spostamento del nodo 6 di $5cm$ per 10 volte. Per valutare se l'errore di stima si accumula lungo i passi viene calcolato l'errore medio di stima per i passi 0, 5 e 10.

3.2.1.1 $\delta = 5cm$

Valutando i risultati riportati in Tabella 19 è possibile osservare che considerando un nodo mobile l'algoritmo riesce a seguire lo sposta-

Tabella 19: Media e deviazione standard dell'errore medio di stima per il nodo 6 con spostamento di 50cm in 10 passi e $\delta = 5$cm.

Media errore medio	0.19
Deviazione standard	0.13

Tabella 20: Media e deviazione standard dell'errore di stima per il nodo 6 nei tre passi valutati con $\delta = 5$cm

	$\bar{\varepsilon}$	σ
Passo 0	0.59	0.33
Passo 5	0.21	0.31
Passo 10	0.13	0.19

mento ma introducendo un errore non indifferente già con un valore di errore della misura di distanza (δ) basso.
Per quanto riguarda l'accumulo di errore tra i passi, Tabella 20, si può osservare che in queste condizioni l'errore non si accumula, ma anzi, diminuisce. L'esempio riportato in Figura 27 conferma le osservazioni appena fatte.
Come ci si aspettava da Figura 28 si vede che tutte le posizioni stimate dei mote subiscono degli spostamenti a causa della variazione delle distanze utilizzate dall'algoritmo.

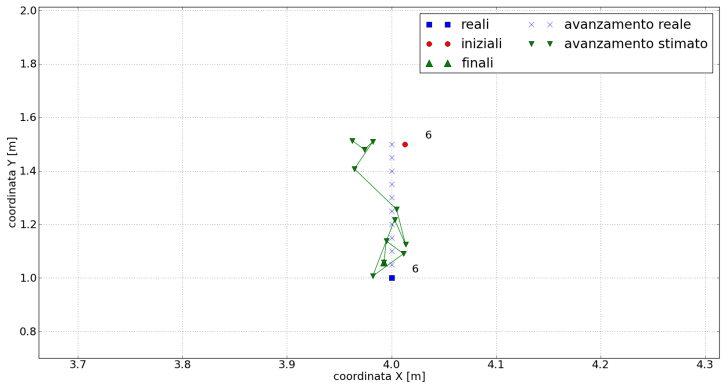


Figura 27: Evoluzione del nodo mobile con 10 passi da 5cm e $\delta = 5$cm

3.2.1.2 $\delta = 10$cm

Questa prova utilizza delle distanze con un errore maggiore; osservando Tabella 21 è subito evidente che l'algoritmo fatica a convergere ad una buona soluzione. Nell'esempio riportato in Figura 29 e Figura 30, oltre a confermare le osservazioni fatte, mostra la fatica dell'algoritmo a mantenere una corretta stima della posizione dei mote che rimangono fermi.

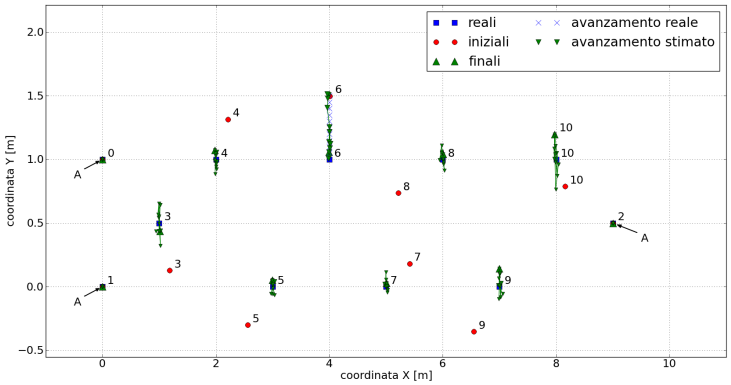


Figura 28: Evoluzione complessiva con 10 passe da 5cm e $\delta = 5cm$

Tabella 21: Media e deviazione standard dell'errore medio di stima per il nodo 6 con spostamento di 50cm in 10passi e $\delta = 10cm$

Media errore medio	0.34
Deviazione standard	0.5

Tabella 22: Media e deviazione standard dell'errore di stima per il nodo 6 nei tre passi valutati con $\delta = 10cm$

	$\bar{\varepsilon}$	σ
Passo 0	0.67	0.35
Passo 5	0.31	0.53
Passo 10	0.37	0.69

Anche in questo caso, come nel caso con $\delta = 5cm$, l'errore non si accumula lungo i passi ma diminuisce.

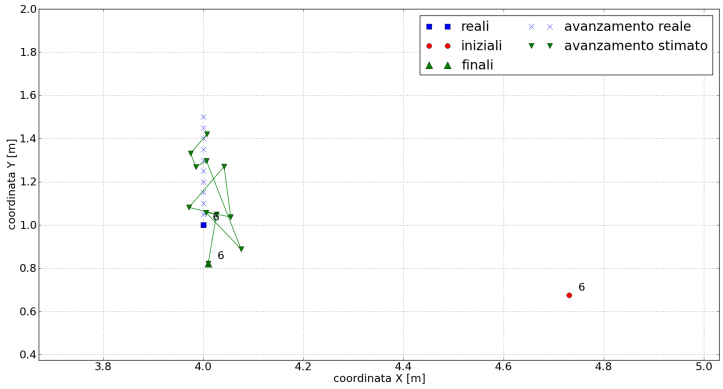


Figura 29: Evoluzione del nodo mobile con 10 passi da 5cm e $\delta = 10cm$

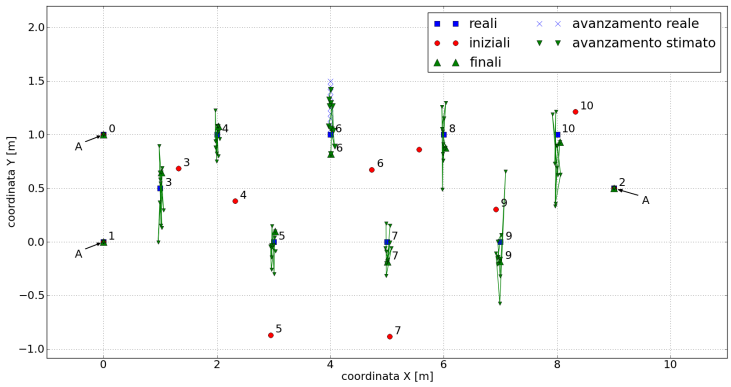


Figura 30: Evoluzione complessiva con 10 passi da 5cm e $\delta = 10cm$

3.2.2 *Configurazione 1: Spostamento del nodo 6 con 5 passi da 10cm*

Nelle prossime due prove il nodo 6 viene spostato della stessa quantità (50cm) ma con meno passi (5) più ampi (10cm).
Per valutare se l'errore di stima si accumula lungo i passi viene calcolato l'errore medio di stima per i passi 0, 3 e 5.

3.2.2.1 $\delta = 5cm$

Come per il caso con 10 passi da 5cm anche in questa prova, utilizzando $\delta = 5cm$ come deviazione standard per l'errore di distanza si ottengono dei discreti risultati nell'inseguimento del nodo mobile.
Per quanto riguarda l'accumulo di errore lungo i passi sono valide considerazioni analoghe alle prove precedenti; infatti anche in questo caso l'errore di stima diminuisce lungo i passi.

Tabella 23: Media e deviazione standard dell'errore medio di stima per il nodo 6 con spostamento di 50cm in 5passi e $\delta = 5cm$

Media errore medio	0.25
Deviazione standard	0.38

Tabella 24: Media e deviazione standard dell'errore di stima per il nodo 6 nei tre passi valutati con $\delta = 5cm$

	$\bar{\varepsilon}$	σ
Passo 0	0.59	0.29
Passo 3	0.18	0.40
Passo 5	0.21	0.52

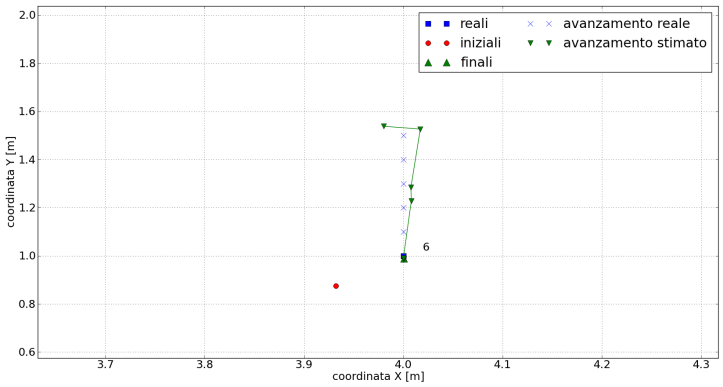


Figura 31: Evoluzione del nodo mobile con 5 passi da 10cm e $\delta = 5cm$

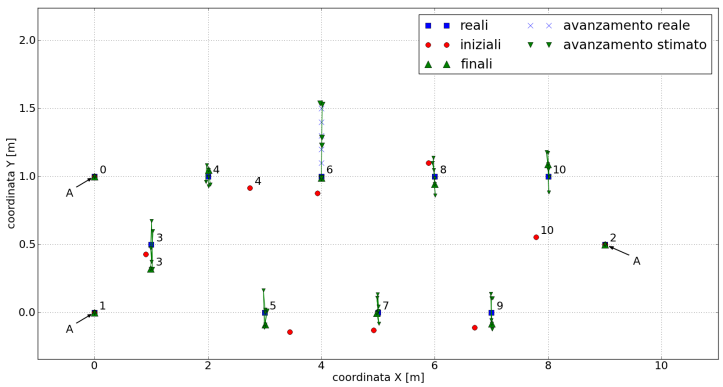


Figura 32: Evoluzione complessiva con 5 passi da 10cm e $\delta = 5cm$

3.2.2.2 $\delta = 10cm$

Tabella 25: Media e deviazione standard dell'errore medio di stima per il nodo 6 con spostamento di 50cm in 5passi e $\delta = 10cm$

Media errore medio	0.36
Deviazione standard	0.27

Tabella 26: Media e deviazione standard dell'errore di stima per il nodo 6 nei tre passi valutati con $\delta = 10cm$

	$\bar{\varepsilon}$	σ
Passo 0	0.65	0.34
Passo 3	0.29	0.33
Passo 5	0.32	0.45

In questa prova non ci sono particolari osservazioni da fare, infatti conferma quanto visto per la corrispettiva prova del caso precedente.

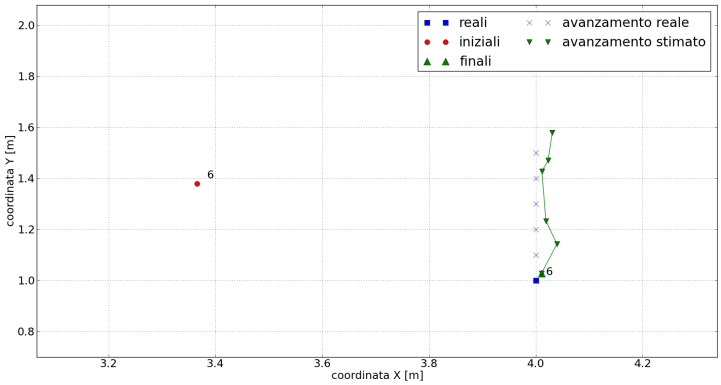


Figura 33: Evoluzione del nodo mobile con 5 passi da 10cm e $\delta = 10cm$

3.2.3 *Conclusioni sulla valutazione sperimentale nel caso dinamico*

Osservando Tabella 27 che riporta i valori, precedentemente commentati, di media e deviazione standard dell'errore di stima per il nodo 6 è immediato accorgersi che il numero e la dimensione dei passi non influenza la stima. Questa osservazione è supportata dai risultati ottenuti alla prima serie di simulazioni, infatti in queste simulazioni si è visto che l'algoritmo riesce a convergere ad una buona stima, con δ uguale a 5 e 10 centimetri, anche con un errore nelle posizioni iniziali alto. Per lo stesso motivo l'errore non si accumula lungo i passi,

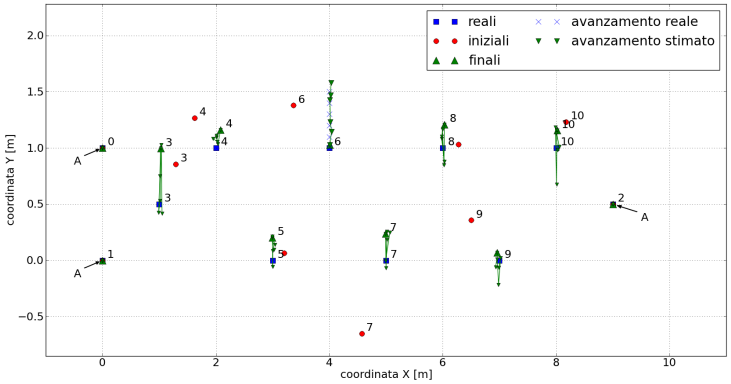


Figura 34: Evoluzione complessiva con 5 passi da 10cm e $\delta = 10cm$

Tabella 27: Tabella riassuntiva con i valori di errore medio e deviazione standard per le due prove.

	$\delta = 5cm$		$\delta = 10cm$	
	$\bar{\varepsilon}$	σ	$\bar{\varepsilon}$	σ
10 passi da 5cm	0.19	0.13	0.34	0.50
5 passi da 10cm	0.25	0.38	0.36	0.79

anzi diminuisce; perchè essendo le condizioni iniziali di ogni passo le stime restituite dall'algoritmo per il passo precedente, queste si avvicinano sempre di più alla posizione reale, diminuendo cos' l'errore nelle condizioni iniziali.

3.3 CONCLUSIONI DELLA VALUTAZIONE SPERIMENTALE

Le simulazioni effettuate in questo capitolo permettono di affermare che l'algoritmo di trilaterazione presentato ha un buon comportamento, sia statico che dinamico, con valori limitati di errore nella misura delle distanze.
In particolare, per la condizione statica si può osservare che se la misura della distanza ha un errore con una deviazione standard fino a 10cm l'algoritmo può essere considerato adatto per la trilaterazione, mentre per la condizione dinamica tale errore porta a risultati meno buoni.
Un errore di questo tipo prevede che il 99.73% delle misure di distanza siano tali che $d - 3\delta < d' < 3\delta + d$ dove d è la distanza esatta, d' è la distanza misurata e δ è la deviazione standard. Quindi con $\delta = 10cm$ si considera che il 99.73% delle misure abbiano un errore inferiore a $\pm 30cm$ rispetto alla distanza esatta.

La scarsa precisione nella stima delle posizioni in condizioni dina-

miche non è un grosso problema in quanto il sistema reale è statico, tranne per qualche circostanza esterna come il passaggio di un animale, fino all'innesco di un *debris flow*. Al verificarsi di questo evento la stima di posizione passa in secondo piano, rispetto alle misure degli accelerometri, perchè in breve tempo il calcolo delle distanze diventerebbe impossibilitato avendo i nodi mobili fuori dalla portata dei nodi ancòra.

4

REALIZZAZIONE SU DISPOSITIVI TMOTE SKY

Per lo sviluppo del sistema reale l'algoritmo validato nel capitolo precedente viene applicato ad una rete di sensori TMOTE^{TM} SKY della *MoteIV* (ora *Sentilla* [1]). La scelta di utilizzare tali dispositivi per lo sviluppo del progetto deriva dalla loro capacità di calcolare la distanza da altri dispositivi, attraverso la potenza del segnale radio ricevuto, con errori compatibili con l'analisi precedente [14]. Inoltre, vista abbondante disponibilità di device già a disposizione del Dipartimento di Ingegneria dell'Informazione dell'Università di Padova, la scelta di utilizzare tali sensori per lo sviluppo del progetto è sembrata corretta.

L'altro componente hardware utilizzato è l'accelerometro ADXL335 della Analog Device scelto per le sue ridotte dimensioni e per il limitato consumo di potenza che lo rendono compatibile con l'applicazione in esame.

Nelle sez. 4.1, 4.2 e 4.3 vengono descritte le caratteristiche hardware e software del sensore Tmote, mentre in sez. 4.4 vengono presentati i risultati sperimentali ottenuti con questo device.
L'aspetto hardware dell'accelerometro viene descritto in sez. 4.5 e la prova effettuata per verificarne il funzionamento in sez. 4.6.

4.1 TMOTE SKY

La piattaforma mote TMOTE SKY è stata progettata insieme al sistema operativo TinyOS da un consorzio guidato dall'Università della California, Berkeley, in collaborazione con *Intel Research* ed è prodotta dalla MoteIV Coporation. Nel 2007 MoteIV ha cambiato nome in Sentilla e ha interrotto la produzione e il supporto per la sua linea di prodotti Tmote, in favore di una nuova piattaforma hardware progettata per le applicazioni Java. Il TMOTE SKY è un modulo wireless a bassissima potenza utilizzato in reti di sensori per applicazioni di monitoraggio e per la realizzazione di prototipi.
Le principali caratteristiche sono:

- Microcontrollore MSP430F1611 della *Texas Instruments* con incorporati 10KB di RAM e 48KB di memoria Flash;
- Memoria flash *ST* M25P80 da 1024kB;
- Chip radio *Chipcon CC*2420 della *Texas Instruments*, compatibile con IEEE 802.15.4 trasmette sui 2.4GHz con una velocità di trasmissione di 250kbps;
- Convertitori ADC e DAC a 12 bit integrati;

- Antenna integrata sul circuito con connettore SMA per antenna esterna (opzionale). L'antenna integrata ha una copertura dichiarata di 50 m all'interno e 125 m all'esterno;
- Consumi di corrente compresi tra 5µA e 21mA;
- Velocità di ripresa dallo stato *sleep* inferiore ai 6µs;
- Sensori di temperatura, umidità e luce integrati;
- Connettore USB utilizzato per programmare il dispositivo e per scambiare dati con il PC;
- Sfrutta lo standard industriale IEEE 802.15.4 per interagire perfettamente con altri dispositivi;
- Connettore di espansione compatibile con I^2C.

La Figura 35 riporta i due lati del modulo, mentre la Figura 36 rappresenta il diagramma a blocchi funzionali.

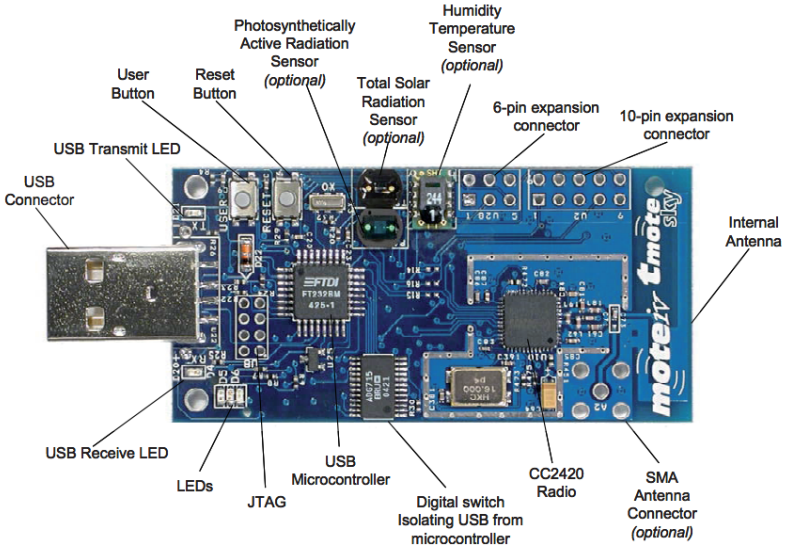
Di seguito verranno descritte con maggior dettaglio alcune caratteristiche dei mote, per informazioni più dettagliate si veda [8].

4.1.1 *Alimentazione*

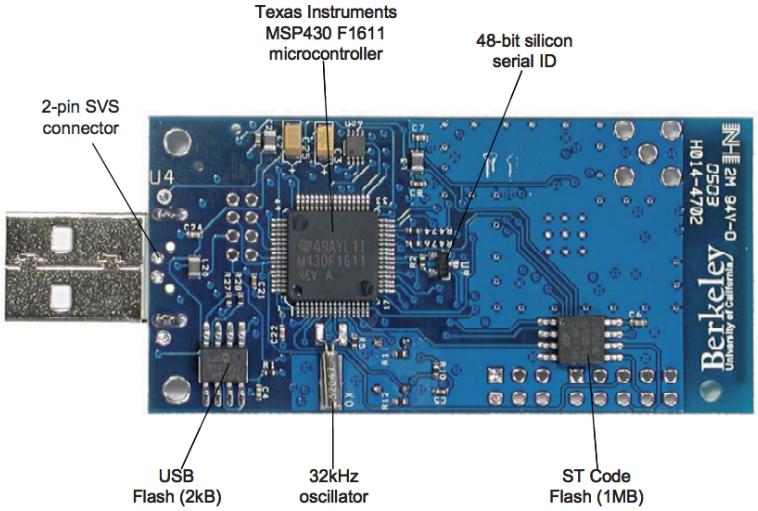
Il modulo Tmote Sky può essere alimentato con due batterie di tipo AA. Il modulo è stato progettato per essere adatto al fattore di forma delle batterie AA. L'alimentazione può essere compresa tra 2.1 e 3.6V DC, tuttavia deve essere di almeno 2.7V quando si programma la memoria flash del microcontrollore e la memoria flash esterna.
Quando il modulo è connesso alla porta USB per la programmazione o la comunicazione con il PC, riceve l'alimentazione da quest'ultimo; in questo caso viene alimentato con 3V. Se il mote è sempre connesso ad una porta USB, il pacco batterie non è necessario.
L'alimentazione può essere fornita anche dai pin 1 e 9 del connettore di espansione U2 descritto in 4.1.5, o dai terminali dedicati alla batteria.
In nessun caso la tensione deve superare i 3.6V per evitare di danneggiare il microcontrollore, la radio, o altri componenti.
La tabella 28 mostra le condizioni operative tipiche del Tmote Sky.

4.1.2 *Microcontrollore*

Il funzionamento a bassa potenza del modulo Tmote Sky è dovuto alla bassissima richiesta di potenza del microcontrollore MSP430F1611, [6], dotato di 10kB di RAM, 48kB di memoria flash, e 128B di memoria per le informazioni. Questo processore con architettura RISC a 16 bit ha consumi di corrente molto bassi sia in fase di lavoro che in fase di *sleep*; ciò permette al mote di funzionare per anni con la stessa



(a) Vista dall'alto



(b) Vista dal basso

Figura 35: Piattaforma TMOTE SKY [8]

Tabella 28: Condizioni tipiche di funzionamento della piattaforma TMOTE SKY

	MIN	NOM	MAX	UNITÁ
Tensione di alimentazione	2.1		3.6	V
Tensione di alimentazione durante la programmazione	2.7		3.6	V
Temperatura di esercizio	−40		85	°C
Consumo di corrente: MCU on, Radio RX		21.8	23	mA
Consumo di corrente: MCU on, Radio TX		19.5	21	mA
Consumo di corrente: MCU on, Radio off		1800	2400	µA
Consumo di corrente: MCU idle, Radio off		54.5	1200	µA
Consumo di corrente: MCU standby		5.1	21.0	µA

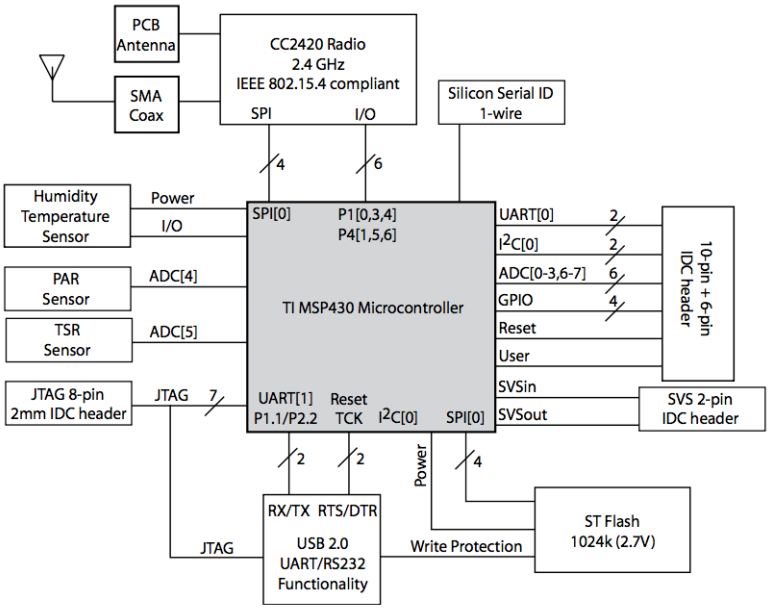


Figura 36: Diagramma a blocchi funzionali del modulo TMOTE SKY.

coppia di batterie.
Il chip possiede un oscillatore interno a controllo digitale (DCO), che può operare fino a 8MHz. Il DCO può essere acceso dalla modalità *sleep* in 6µs, tuttavia a temperatura ambiente il tempo tipico di accensione è di 292ns. Quando il DCO è spento il processore utilizza un clock generato da un cristallo di quarzo da orologi da 32768Hz.
Oltre al DCO, l'MSP430 ha 8 porte ADC esterne e 8 interne. Queste ultime possono essere utilizzate per leggere il termistore interno o monitorare il livello di tensione delle batterie. Possiede inoltre una varietà di periferiche, tra cui bus SPI, porta UART, porte I/O digitali, timer *Watchdog* (per l'auto-reset del dispositivo in caso di *loop* infinito), timer con funzioni di cattura e confronto, 2 porte DAC a 12 bit, un *supply voltage supervisor* e un controllore DMA a 3 porte.

4.1.3 *Comunicazione con il PC e programmazione*

I TMOTE SKY utilizzano un *controller* della FTDI per comunicare con il PC, utilizzando l'interfaccia UART tramite la porta USART1 del microprocessore.
Il mote ha un speciale circuito hardware che impedisce che ci siano dei reset spuri della memoria flash, per questo è necessario un particolare software che in fase di programmazione invii una specifica sequenza per poter cancellare la memoria flash.

4.1.4 *Flash esterna*

La memoria E²PROM presente sul modulo è la M25P80 della STMicroelettronics, ed è utilizzata per memorizzare il codice e i dati esterni. Questo chip contiene 1024kB divisi in 16 segmenti da 64kB ciascuno. La memoria condivide con il ricetrasmettitore CC2420, sez. 4.1.6, la linea di comunicazione SPI; di conseguenza bisogna prestare attenzione quando le letture e le scritture della memoria flash sono intervallate da comunicazioni radio, perché non possono avvenire contemporaneamente.

4.1.5 *Connettore di espansione*

Il TMOTE SKY ha due connettori di espansione.
Sul lato opposto al connettore USB, in posizione U2 ed U28 (Figura 35a), si trovano rispettivamente un connettore IDC a 10 pin ed uno a 6 pin. Il connettore a 10 pin è quello principale; esso fornisce l'ingresso e l'uscita di segnali digitali e analogici. Le periferiche possono essere collegate utilizzando un cavo *flat* o progettando un circuito stampato che può essere saldato direttamente al connettore, fornendo un collegamento robusto.
L'altro connettore (U28) fornisce due ulteriori ingressi analogici che se opportunamente riconfigurati via software possono essere usati come due uscite DAC da 12 bit. Inoltre l'ADC7 (presente nel connettore U28) può anche agire come supervisore del voltaggio di alimentazione, mentre gli elementi dell'interfaccia utente (i pulsanti *user* e *reset*) vengono esportati dal connettore per facilitarne l'utilizzo e il *packaging*.

4.1.6 *Radio*

Il chip radio utilizzato dal TMOTE SKY per le comunicazioni wireless è il CC2420, a basso consumo, basso voltaggio e basso costo, della serie Chipcon della Texas Instruments [11]. Questo è il componente di maggiore interesse per lo scopo di questo lavoro.
Il CC2420 è conforme alle norme IEEE 802.15.4 e fornisce il livello fisico e alcune funzioni del livello MAC [13]. Il chip è totalmente programmabile, infatti si può settare, sia in fase di compilazione che in real-time, il valore della potenza trasmessa e il canale di trasmissione.
La potenza di trasmissione può essere impostata scegliendo tra 32 livelli (da 0 a 31) ad ognuno dei quali corrisponde un valore di potenza e un consumo di corrente. Alcune di queste corrispondenze sono riportate il tabella 29. Per quanto riguarda il canale di trasmissione poiché il chip lavora nella banda non licenziata intorno ai 2.4GHz per lo standard IEEE 802.15.4 [13] si può scegliere tra 16 canali, da 11 a

Tabella 29: Configurazioni della potenza di uscita per il CC2420

PA LEVEL	Registro TXCTRL	Potenza d'uscita [dBm]	Consumo di corrente [mA]
31	0xA0FF	0	17.4
27	0xA0FB	−1	16.5
23	0xA0F7	−3	15.2
19	0xA0F3	−5	13.9
15	0xA0EF	−7	12.5
11	0xA0EB	−10	11.2
7	0xA0E7	−15	9.9
3	0xA0E3	−25	8.5

26.

Il chip CC2420 fornisce inoltre un indicatore di potenza del segnale ricevuto, RSSI (*Received Signal Strenght Indicator*), che può essere letto in qualsiasi momento. Questo indice è quantizzato su un insieme di 100 valori e possiede un'accuratezza di 6[dBm][1]. Inoltre, su ogni pacchetto ricevuto il CC2420 calcola il tasso di errore e produce una indicazione della qualità della connessione, LQI (*Link Quality Indicator*). Entrambi gli indici vengono salvati in una variabile a 8 bit, con la differenza che il primo viene salvato con segno, quindi assume valori tra −127 e 128; mentre il secondo viene salvato senza segno, quindi assume valori tra 0 e 255.
La Figura 37 riporta la mappatura dell'indice RSSI in funzione della potenza del segnale ricevuto.

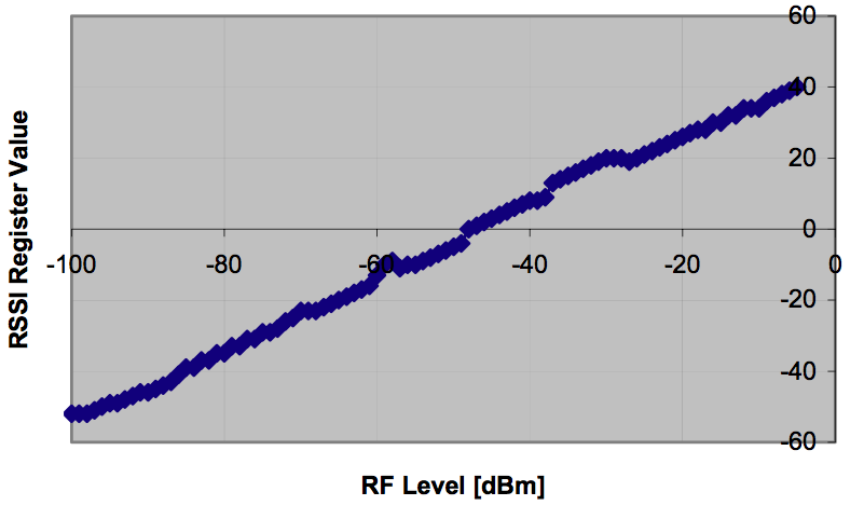


Figura 37: Mappatura dell'indice RSSI in funzione della potenza del segnale ricevuto [8].

1 Si definisce una generica potenza P espressa in [dBm] come $P_{[dBm]} = 10 \log \frac{P[mW]}{1[mW]}$.

L'antenna utilizzata dal mote è un'antenna a F-invertita stampata sulla scheda. Sebbene non abbia una radiazione perfettamente omnidirezionale, l'antenna raggiunge i 50m all'interno e i 125m all'aperto. In Figura 38 vengono riportati i diagrammi di radiazione dell'antenna, quando il mote si trova in posizione verticale e orizzontale.

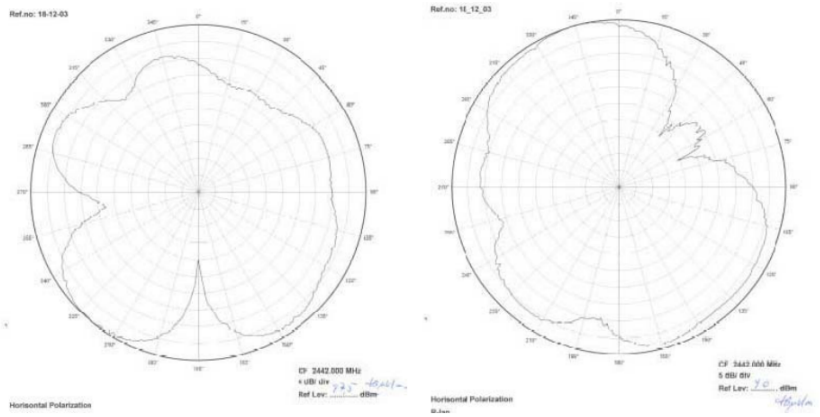


Figura 38: Diagramma di radiazione dell'antenna ad F-invertita del TMOTE SKY [8].

4.2 SISTEMA TINYOS

TinyOS è un sistema operativo open-source progettato per reti di sensori wireless. La prima versione del sistema (v0.43) è stata rilasciata nel 2000, mentre l'ultima (v2.1) è del 2008. TinyOS è stato originariamente sviluppato come progetto di ricerca dell'Università della California Berkeley, ma la sua popolarità è cresciuta fino ad avere una comunità internazionale di sviluppatori e utilizzatori.

Nelle tradizionali architetture hardware ci si trova a disporre di grandi quantitativi di risorse, come la memoria, la capacità computazionale e l'illimitata fonte di energia. Tale disponibilità di risorse viene a mancare quando si considerano reti di sensori, dove ci sono problemi come le ridotte dimensioni, la scarsa quantità di memoria, le modeste capacità di elaborazione e soprattutto le fonti di energia limitate. Questi problemi necessitano di soluzioni efficienti in particolare per ridurre al massimo i consumi di energia. Il sistema operativo TinyOS non possiede un nucleo e permette l'accesso diretto all'hardware. Per evitare lo spreco di energia causato dall'overhead, che nelle reti di sensori non porta vantaggi sufficienti a giustificarlo, spariscono i concetti di processore virtuale e memoria virtuale; quest'ultima viene considerata come un unico e lineare spazio fisico assegnato alle applicazioni a tempo di compilazione.

Dunque TinyOS deve sottostare ai seguenti requisiti:

- ridurre i consumi di energia;
- ridurre il carico computazionale e le dimensioni del sistema operativo;
- supportare richieste di operazioni che devono essere svolte in concorrenza e in maniera tale da raggiungere un alto livello di robustezza ed un'efficiente modularità.

Il requisito della modularità è garantito dal fatto che TinyOS è composto da un grande numero di piccoli componenti riutilizzabili; la riusabilità di ogni componente, ed una sua eventuale sostituzione, è permessa dalla definizione di un'interfaccia. Oltre a ciò l'ingegneria del software a componenti incoraggia l'uso di schemi di architettura prevedibili, e di un'infrastruttura software standard, portando così a risultati di qualità superiore [10]. La struttura così articolata di TinyOS fa si che quando viene installata un'applicazione su un sensore il sistema operativo viene installato solo con i componenti necessari all'applicazione stessa, riducendo così lo spazio di memoria occupato e soddisfando un ulteriore requisito. Infatti il software che viene installato su un sensore occupa pochissima memoria. Oltre che per piattaforme con memoria ridotta, TinyOS è stato progettato appositamente per rispettare tutti i vincoli sulle risorse dei sensori wireless, in primo luogo la scarsa disponibilità di energetica.

TinyOS è stato realizzato secondo il modello ad eventi. Questo modello si confà molto bene alle necessità di un sistema con memoria ridotta e sottoposto a continue sollecitazioni. Infatti i nodi non dispongono dei requisiti necessari per lavorare come nei modelli tradizionali, nei quali viene allocata della memoria nello stack per ogni attività obbligando il sistema a frequenti commutazioni di contesto.

Nel campo delle reti di sensori, il consumo di energia è un fattore critico e l'approccio basato sugli eventi utilizza il microprocessore nella maniera più efficiente possibile. Quando il sistema viene sollecitato da un evento, questo viene gestito immediatamente e rapidamente. In effetti non sono permesse condizioni di blocco, né attese attive che sprecherebbero energia inutilmente. Quando invece non ci sono attività da eseguire, il sistema mette a riposo il microprocessore che viene risvegliato all'arrivo di un evento.

4.2.1 *Componenti*

Il sistema operativo TinyOS è sviluppato in nesC, descritto in 4.3, ed è composto da un insieme di librerie e applicazioni che possono essere utilizzate per sviluppare ulteriori applicazioni. A loro volta le librerie e le applicazioni sono costituite da componenti.

Ogni componente presente in TinyOS è un'unità software indipendente dotata di interfacce che svolge un determinato compito. L'insieme dei componenti che formano un'applicazione si chiama *grafo dei*

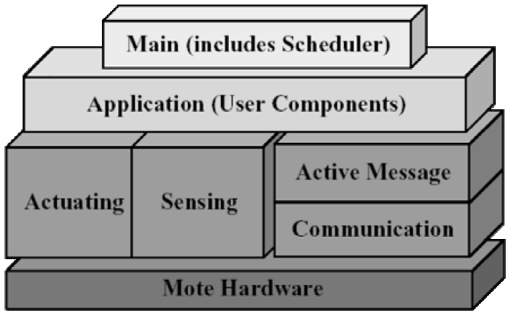


Figura 39: Architettura a componenti di un'applicazione TinyOS

componenti. Questo fissa una gerarchia tra i componenti stessi, quindi si ha che un componente di livello inferiore notifica eventi (*event*) ai componenti di livello più alto, mentre i componenti di livello superiore richiedono servizi (*command*) ai componenti di livello più basso. Come si vede in Figura 39 i componenti che formano l'architettura di un'applicazione si possono dividere in tre categorie:

- Lo strato più basso è formato da componenti che rappresentano direttamente i dispositivi hardware creando un'astrazione software di questi. Questi componenti si chiamano **hardware abstraction**.
  Un componente che legge o scrive un bit su un canale radio, per esempio, fa parte di questa categoria.

- Lo stato intermedio è formato da componenti, chiamati **synthetic hardware**, che simulano il comportamento di dispositivi hardware più complessi e sofisticati di quelli realmente a disposizione.
  Per esempio, un componente che prende 8 bit forma un byte per passarlo al livello superiore.

- Il livello più alto (le applicazioni dell'utente) è formato da componenti che eseguono algoritmi che si astraggono dai dispositivi hardware e software. Questi vengono chiamati **high-level software component**; ne fa parte, per esempio, un componente che implementa il protocollo di routing.

TinyOS non utilizza il tradizionale modello basato su thread, bensì un modello di programmazione basato su macchine a stati. Infatti ogni componente transita da uno stato all'altro in seguito ad una richiesta di operazione o in seguito alla notifica dell'arrivo di un evento. Ogni transizione avviene in modo praticamente istantaneo e non esiste nessun evento che possa interromperla.
Questo comportamento dei componenti è dovuto alla struttura del componente stesso, che è diviso in **comandi**, **eventi**, **frame** e **task**.

Il **frame** è l'area dati del componente, è allocata a tempo di compilazione e permette di conoscere la quantità di memoria richiesta dal-

l'intera applicazione. Questa soluzione permette di evitare gli sprechi associati all'overhead per l'allocazione di memoria.

I **comandi** sono delle richieste per servizi forniti da un componente di livello inferiore. Ritornano un valore che indica se la chiamata è andata a buon fine.
Si possono distinguere due tipi di comandi. Il primo tipo effettua la richiesta depositando i parametri nel frame e poi attiva un task. Mentre il secondo dopo aver depositato i parametri chiama un altro comando.

Gli **eventi** prevedono l'intervento di due tipologie di componenti; i *supplier* che producono gli eventi, e i *consumer* che ricevono gli eventi. Questi possono interagire in due versi, formando due modelli di notifica degli eventi:

PUSH MODEL Il componente supplier notifica l'evento al consumer di sua iniziativa;

PULL MODEL Il componente consumer richiede l'evento al supplier.

Come per i comandi, anche un gestore di evento può depositare dei parametri nel frame e poi avviare un task.
Gli eventi non possono essere notificati dai comandi, questo per evitare che si possano verificare dei loop infiniti. Infatti gli eventi possono notificare altri eventi e chiamare dei comandi, quindi se un comando potesse notificare un evento, il quale chiama il comando stesso ci si troverebbe in una situazione di stallo.

Il **task** è l'attività unica ed indipendente di cui è composto il programma in esecuzione.
I task non hanno diritto di prelazione su nessuna attività, quindi sono *nonpreemptive*. Essi possono chiamare comandi, invocare o notificare eventi e attivare altri task all'interno dello stesso componente. Hanno il compito di eseguire grandi quantità di calcoli che non siano critici temporalmente.

Il componente principale per ogni applicazione, quello che è sempre presente, è lo **scheduler**. Questo componente si occupa di mandare in esecuzione i task secondo una politica FIFO ("First-In-First-Out"). Ha due livelli di priorità, uno per i task e uno più per gli eventi, che possono interrompere l'esecuzione di un task.
Se la coda dei task è vuota lo scheduler mette a riposo il microprocessore, per evitare inutili sprechi di energia.

4.2.2 *Split-phase operation*

In TinyOS la comunicazione tra componenti situati a livelli diversi avviene con operazioni *Split-phase*, cioè in modo asincrono con accoppiamento lasco[2]. Il meccanismo utilizzato sfrutta il modello ad eventi

2 Accoppiamento lasco: Consente l'evoluzione di processi in modo indipendente con interazioni limitate

per superare i limiti introdotti dal *distributed callback*, sul quale il sistema operativo si basa.
Infatti il *distributed callback* è una tecnica di comunicazione asincrona con la quale il cliente invia una richiesta al servente e prosegue nella propria elaborazione. Il servente, eseguito il servizio, interrompe il cliente richiamando un'apposita operazione del cliente stesso per la restituzione dei risultati. Il limite di questa tecnica è il legame stretto presente tra cliente e servente che permette una comunicazione uno ad uno, impedendo la realizzazione di un meccanismo di comunicazione tra un gruppo di clienti ed un gruppo di serventi.

In TinyOS, quindi, ogni operazione non banale diventa un operazione *split-phase*, è cioè divisa in due funzioni, la chiamata e il ritorno dei risultati. Quando un componente effettua una chiamata di un comando su un componente di livello inferiore, il comando avvia un task in questo componente e restituisce subito lo stato della chiamata al componente chiamante. Quando il task chiamato nel componente di livello inferiore termina viene notificato un evento con i risultati al componente chiamante.
In Figura 40 è riportata la semantica di quanto appena descritto, dove *ComponentA* è il componente chiamante, e *ComponentB* è il componente che offre il servizio richiesto. I due componenti coinvolti nell'ope-

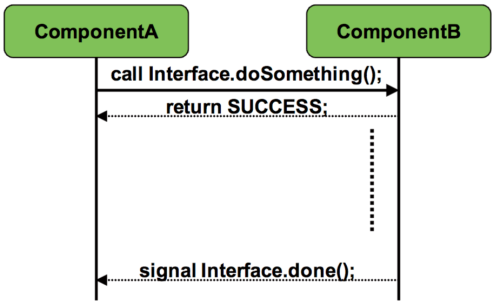


Figura 40: Semantica split-phase.

razione devono implementare le due parti dell'operazione stessa; in particolare, il componente chiamato deve implementare il comando e il componente chiamante deve implementare l'evento.

4.2.3 *Active messages*

L'active messages è una tecnologia molto leggera che implementa il livello *data-link* del protocollo TCP/IP controllando l'accesso al mezzo trasmissivo e la comunicazione single-hop tra i nodi della rete.
Essa permette la comunicazione tra i nodi se questi sono collegati dal mezzo trasmissivo. Infatti le funzioni di routing vengono implementate a livello superiore.
L'active messages non specifica particolari meccanismi di comunica-

zione e prevede che ogni pacchetto sia un'entità indipendente. Questo pacchetto contiene almeno 9 byte così divisi, e con questi significati:

- 1° byte: serve per indicare che il pacchetto è di tipo active messages;
- 2°-3° byte: indirizzo di destinazione del messaggio, se posto a 0xFFF il messaggio viene spedito in broadcast;
- 4°-5° byte: indirizzo del mittente del messaggio;
- 6° byte: lunghezza in byte del messaggio vero e proprio;
- 7° byte: ID del gruppo. Questo campo permette di mandare il messaggio ad un gruppo ristretto di mote, potendo così suddividere una grande rete in tante sottoreti logicamente separate e invisibili tra di loro;
- 8° byte: tipo di messaggio. Questo byte indica l'evento che deve essere notificato dall'arrivo del messaggio. Quando un pacchetto viene ricevuto da un nodo, l'*active message* lo elabora, incapsula il campo dati del pacchetto nell'evento, ed invia la notifica dell'evento al componente appropriato. Questa caratteristica permette a più reti o a più protocolli di alto livello di funzionare in concorrenza;
- 9° byte: messaggio, può essere lungo al massimo 28°.

4.3 LINGUAGGIO NESC

Il linguaggio nesC è una variante del linguaggio C. Per alcuni aspetti può considerarsi un'estensione, in quanto implementa un modello ad eventi; per altri aspetti è considerata una limitazione, infatti per ragioni di maggior efficienza e semplicità limita l'utilizzo dei puntatori e non permette l'allocazione della memoria in modo dinamico.

Per sviluppare un'applicazione in nesC è necessario realizzare tutti i componenti che vengono poi assemblati per formare il codice eseguibile.
Ogni componente deve definire le specifiche del suo comportamento e implementare tali specifiche. Per fare ciò viene utilizzata una serie di interfacce. Queste possono definire le funzionalità fornite da altri componenti che si intendono utilizzare per realizzare il proprio comportamento, o definire le funzionalità che si mettono a disposizione ad altri componenti.
Ognuna di esse specifica una serie di funzioni che devono essere implementate da chi fornisce l'interfaccia, ed una serie di funzioni che devono essere implementate dal componente che utilizza l'interfaccia. Questo garantisce che non ci siano operazioni bloccanti, nemmeno se

richiedono un tempo di esecuzione non determinato, come ad esempio l'invio di pacchetti. Infatti un componente che utilizza l'interfaccia per l'invio dei pacchetti non può utilizzare il comando *send* (invio del pacchetto) senza implementare l'evento *send_done* (invio del pacchetto completato).
Le interfacce legano tra loro i componenti in modo statico. Questo per ottenere una maggiore robustezza, aumentare la velocità di esecuzione e per permettere l'analisi statica del programma. Infatti, nel caso ci fossero problemi legati all'accesso concorrente di una risorsa, il compilatore sarebbe in grado di accorgersene.

Il sistema operativo TinyOS è scritto in nesC, e quando un applicazione viene compilata i componenti che la costituiscono vengono compilati insieme ai componenti che formano TinyOS; il risultato finale è l'intero software del sensore. Questa prassi permette di risparmiare memoria ed energia, però ha lo svantaggio di non essere particolarmente versatile poiché non consente di installare due applicazioni sullo stesso nodo sensore e non prevede la possibilità di effettuare linking dinamico di librerie esterne.

4.3.1 *Moduli e configurazioni*

I componenti di nesC possono essere di due tipi: *moduli* o *configurazioni*.

Un modulo è un componente che implementa le interfacce che esso stesso dichiara, definendo i comandi delle interfacce pubblicate, e gli eventi delle interfacce utilizzate. Per notificare un evento viene utilizzata la parola chiave `signal`, mentre per chiamare un comando la parola chiave da utilizzare è **`call`**. Ogni modulo contiene il suo stato sotto forma di variabili, definite in modo analogo al C. Qui di seguito viene riportato un esempio di modulo nel quale vengono definite le interfacce utilizzate: 3 timer, la gestione dei led e l'interfaccia *boot*; e vengono implementati gli eventi forniti da queste interfacce, in particolar modo: l'evento *booted* avvia i timer in modo periodico definendo il periodo in millisecondi e gli eventi relativi all'esaurirsi di un periodo di timer fanno lampeggiare un led.

```
module BlinkC {
  uses interface Timer<TMilli> as Timer0;
  uses interface Timer<TMilli> as Timer1;
  uses interface Timer<TMilli> as Timer2;
  uses interface Leds;
  uses interface Boot;
}
implementation
{
  event void Boot.booted()
  {
    call Timer0.startPeriodic( 250 );
```

```
    call Timer1.startPeriodic( 500 );
    call Timer2.startPeriodic( 1000 );
  }

  event void Timer0.fired()
  {
    call Leds.led0Toggle();
  }

  event void Timer1.fired()
  {
    call Leds.led1Toggle();
  }

  event void Timer2.fired()
  {
    call Leds.led2Toggle();
  }
}
```

Il risultato finale è la visualizzazione dei 3 bit meno significativi di un contatore incrementato ogni quarto di secondo.

Una configurazione è un componente che si occupa di collegare tra loro altri componenti e di rinominare ed esportare interfacce. Ossia, collegare le interfacce utilizzate da alcuni componenti con le interfacce pubblicate da altri componenti, collegare un'interfaccia utilizzata dalla configurazione con un'interfaccia pubblicata dalla configurazione e delegare ad altri componenti l'implementazione di interfacce dichiarate dalla configurazione.

Ogni applicazione deve avere una configurazione globale che definisce i legami tra tutti i componenti di cui l'applicazione è composta. Questa configurazione deve contenere una sezione di implementazione, nella quale definisce tutti i componenti utilizzati tramite il comando **components** e specifica i collegamenti tra questi componenti tramite i *vincoli di wiring*. Tali vincoli vengono specificati utilizzando l'operatore $->$. La sintassi vuole che la direzione della freccia vada dall'interfaccia utilizzata verso l'interfaccia pubblicata.

Di seguito viene riportato un esempio di configurazione:

```
configuration BlinkAppC {
}
implementation {
  components MainC;
  components LedsC;
  components BlinkC as App;
  components new TimerMilliC() as Timer0;
  components new TimerMilliC() as Timer1;
  components new TimerMilliC() as Timer2;

  App.Boot -> MainC;
  App.Leds -> LedsC;
```

```
  App.Timer0 -> Timer0;
14  App.Timer1 -> Timer1;
  App.Timer2 -> Timer2;
16 }
```

Questa specifica configurazione serve per definire i legami necessari per utilizzare il modulo riportato nell'esempio precedente. In questa configurazione dopo aver definito i componenti utilizzati, assegnando ad essi anche dei nominativi per poter utilizzare più istanze dello stesso componente, vengono specificati i collegamenti tra le interfacce. Per esempio viene collegata l'interfaccia *Leds* utilizzata nel componente *App*, che sarebbe *BlinkC* (quello riportato nell'esempio precedente), con l'interfaccia *Leds* pubblicata dal componente *LedsC*.

4.3.2 *Assemblaggio dei componenti*

Come già spiegato, per realizzare un applicazione è necessario collegare tra loro i componenti che la formano. In Figura 41 è riportato un esempio di grafo dei componenti. Nell'esempio i componenti *Compo-*

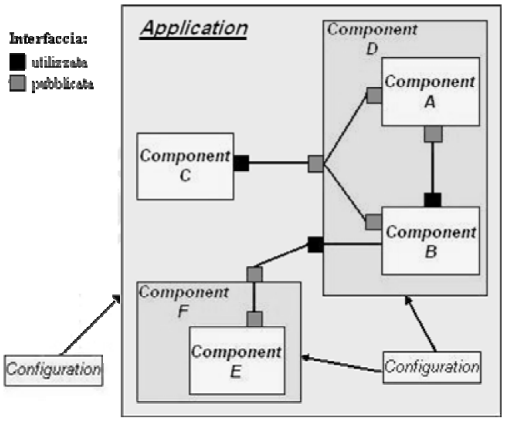


Figura 41: Esempio di grafo dei componenti

nentD, *ComponentF* e *Application* sono delle configurazioni, infatti collegano tra loro gli altri componenti preesistenti. Per esempio, il *ComponentD* fornisce un'interfaccia, che verrà utilizzata da *ComponentC*, e utilizza un'interfaccia che viene fornita da *ComponentF*.

4.3.3 *Stratificazione dei componenti*

La Figura 42 mostra un esempio di stratificazione di un'applicazione. Ciò che maggiormente interessa è la relazione che intercorre tra i vari strati di un'applicazione. Questa relazione favorisce la modularità, infatti volendo modificare il protocollo di routing basta modificare solo i componenti appartenenti al blocco *Routing layer*. Un discorso analogo è valido se si vuole modificare la piattaforma hardware, in quanto

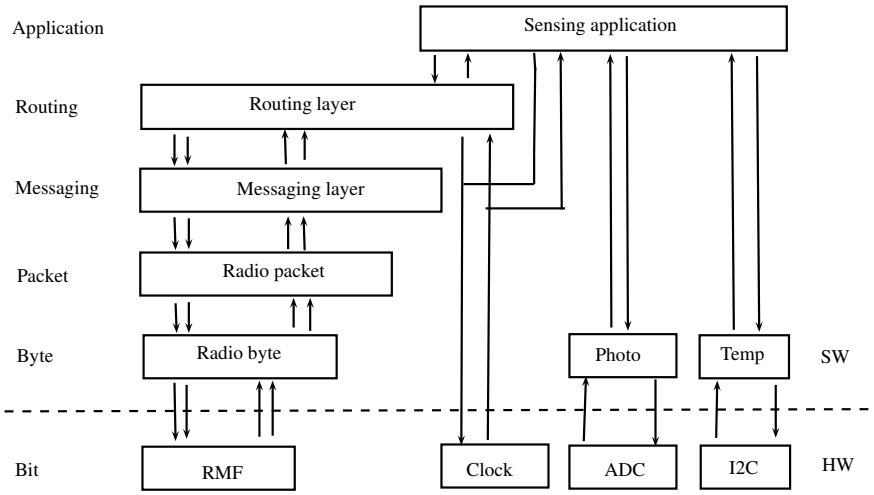


Figura 42: Stratificazione di un applicazione

un interrupt hardware viene tradotto in un evento software dai blocchi più bassi. Quindi se si vuole sostituire il sistema di comunicazione è sufficiente modificare i componenti del blocco *RMF*.

4.3.4 *Modello concorrente*

Un programma nesC non prevede una esecuzione lineare, ma un esecuzione divisa in task ed interruzioni hardware. I task possono essere attivati da comandi, eventi o altri task tramite la parola chiave **`post`** e vengono implementati tramite una funzione senza argomenti e senza valore di ritorno preceduta dalla parola chiave **`task`**.
I task non possono essere interrotti da altri task, mentre possono essere interrotti da interrupt hardware. Si possono considerare due tipi di codice: codice *sincrono* che può essere mandato in esecuzione solo da un task e codice *asincrono* che può essere mandato in esecuzione solo da un'interruzione hardware. Il codice sincrono è eseguito in modalità run-to-completion, quindi non può essere interrotto da altro codice sincrono. Il codice asincrono, invece, può interrompere il codice sincrono in esecuzione. In altre parole, il codice sincrono è atomico rispetto ad altro codice sincrono e non lo è rispetto a codice asincrono.

La politica run-to-completion elimina i problemi relativi all'uso di risorse condivise all'interno di pezzi di codice sincrono. Il problema rimane però se le risorse vengono utilizzate anche da pezzi di codice asincrono. Per risolvere questo problema il compilatore controlla sempre (ciò è permesso dal linking dinamico, dalla mancanza di allocazione di memoria in modo dinamico e dalle restrizioni sull'uso dei puntatori) che le modifiche ad un elemento dello stato condiviso avvenga sempre in sezioni di codice sincrono o in sezioni atomiche.

Una sezione atomica è una porzione di codice eseguita disabilitan-

do gli interrupt hardware. Queste sezioni vengono delimitate utilizzando la parola chiave **`atomic`** e non permettono di invocare comandi o notificare eventi per garantire che l'esecuzione della sezione non sia troppo lunga.

4.4 MODELLO DEL CANALE

Per la stima della distanza tra due nodi è utilizzato il valore di RSS (*Receive Signal Strength*), ottenuto a partire dal valore di RSSI (*Receive Signal Strength Indicator* ritornato direttamente dal chip radio montato sui TMOTE SKY. Questi due valori, secondo quanto riportato in [11] sono legati dalla seguente relazione:

$$\mathrm{RSS} = \mathrm{RSSI} - 45$$

Per poter risalire alla distanza a partire dal valore di RSS è indispensabile considerare il modello di canale in cui si va ad operare.
Le misure di potenza sono affette da due tipi di errori: errori statici ed errori variabili nel tempo. I primi dipendono da attenuazioni, riflessioni, diffrazioni e diffusioni dovute alla presenza di oggetti nell'ambiente in cui si effettuano le misure; i secondi sono causati da rumore generico ed interferenze.
Il modello completo del canale in termini di potenza ricevuta P_{rx} [dBm] può essere modellizzato nel seguente modo, [14]:

$$P_{rx}^{ij} = P_{tx}^{j} + r_j + f_{pl}\left(\|x_i - x_j\|\right) + f_{sf}(x_i, x_j) + f_a(x_i, x_j) + \nu_{ff}(t) + o_i$$

dove i e j sono rispettivamente il nodo ricevente e il nodo trasmittente, $x_i, x_j \in \mathbb{R}^3$ sono le posizioni nello spazio tridimensionale e t è l'istante in cui avviene la comunicazione. P_{tx}^{j} è la potenza di trasmissione, r_j è l'offset di trasmissione, $f_{pl}(\cdot)$ è l'effetto di *path loss*, $f_{sf}(\cdot)$ rappresenta lo *slow fading*, $f_a(\cdot)$ è il fattore di asimmetria del canale, $\nu_{ff}(\cdot)$ costituisce la componente di *fast fading* e $o_i(\cdot)$ è l'offset sulla misura dell'intensità ricevuta.
Da questo modello completo è possibile, attraverso alcuni accorgimenti, eliminare o semplificare dei parametri difficilmente ricavabili, giungendo ad un modello approssimato. In particolar modo:

P_{tx}^{j} E r_j Rappresentano rispettivamente la potenza di trasmissione in dBm e la differenza tra la potenza nominale da datasheet e quella effettivamente trasmessa a causa di difetti di fabbricazione. Si presumono costanti nel tempo e sono circa uguali a 0 se i mote vengono utilizzati con il livello di potenza di trasmissione massimo (Tabella 29);

$f_{pl}(\cdot)$ Il *path loss* è definito come la riduzione della densità di potenza di un'onda elettromagnetica al propagarsi nello spazio. Il suo effetto si può modellizzare come [14]:

$$f_{pl}\left(\|x_i - x_j\|\right) = \beta - 10\gamma \log_{10}\left(\|x_i - x_j\|\right)$$

dove β è il guadagno del ricevitore radio alla distanzi di $1m$ e γ è il fattore di perdita.

$f_{sf}(\cdot)$ Lo *slow fading* è un fenomeno lentamente variabile che tiene conto dalla variazione morfologica dell'ambiente circostante; può essere causato da fenomeni di *shadowing*, dove un ostacolo voluminoso, come una collina od un edificio, oscura il percorso del segnale tra trasmettitore e ricevitore. Di conseguenza se il modello del canale viene calcolato sul luogo di utilizzo del sistema questo parametro non varia.

$f_a(\cdot)$ L'asimmetria del canale è dovuta a riflessioni non simmetriche ed è modellizzabile come una variabile gaussiana a media nulla.

$\nu_{ff}(\cdot)$ Il *fast fading* è un fenomeno che provoca rapide fluttuazioni del segnale ricevuto. Mediando un insieme di N misure di potenza ricevute dal nodo i rispetto al nodo j in N istanti ravvicinati, $\hat{P}_{rx}^{ij} = \sum_{k=1}^{N} P_{rx}^{ij}(k)$, viene rimosso il suo effetto.

$o_i(\cdot)$ L'offset del nodo ricevente è dovuto a difetti di fabbricazione del chip radio. Questo termine si può considerare trascurabile.

Per maggiori informazioni riguardo ai parametri del modello completo del canale si faccia riferimento a [14].
Applicando queste semplificazioni al modello completo del canale si ottiene che la potenza media ricevuta diviene:

$$\hat{P}_{rx}^{ij} = \beta - 10\gamma \log_{10}\left(||x_i - x_j||\right) + f_a(x_i, x_j)$$

di conseguenza è necessario stimare solo i parametri β e γ ed essendo la componente di asimmetria del canale una variabile aleatoria gaussiana è possibile utilizzare uno stimatore ai minimi quadrati.

Si prenda una coppia di nodi (i, j) e si effettuino un totale di $N = \sum_{k=1}^{d} N_k$ misure di P_{rx}^{ij} con $d \in \mathbb{N}$, ottenute tenendo fermo il nodo i e spostando il nodo j in d distanze diverse, alle quali vengono raccolti gli N_k, con $k = 1, \ldots, d$ campioni. Si definisce dunque il modello di regressione lineare:

$$\mathbf{b} = \begin{bmatrix} b_1 \\ \vdots \\ b_N \end{bmatrix} = A \begin{bmatrix} \beta \\ \gamma \end{bmatrix} + \begin{bmatrix} w_1 \\ \vdots \\ w_N \end{bmatrix} = A\theta + \mathbf{w}$$

$$\text{con} \quad \mathbf{b} = \begin{bmatrix} P_{rx_1}^{ij} \\ \vdots \\ P_{rx_N}^{ij} \end{bmatrix} \quad e \quad A = \begin{bmatrix} 1 & -10\log_{10}\left(||x_i - x_j||_1\right) \\ \vdots & \\ 1 & -10\log_{10}\left(||x_i - x_j||_N\right) \end{bmatrix} \tag{8}$$

dove il vettore aleatorio $\mathbf{w}$ è a media nulla e varianza σ_a^2; sotto questa ipotesi si dimostra [12] che lo stimatore ai minimi quadrati risulta:

$$\hat{\theta}_{LS} = (A^T A)^{-1} A^T \mathbf{b} \tag{9}$$

4.4.1 *Calcolo del modello del canale*

In questo lavoro di tesi, diversamente da quanto fatto in progetti precedenti [14] in cui viene calcolato un modello di canale unico per tutti i mote, il modello è calcolato per ogni coppia ordinata di mote in modo da avere una maggiore precisione sulla stima della distanza tra i nodi.
Per non appesantire il codice dei sensori con il calcolo del modello del canale questa operazione viene eseguita dal PC. Questo porta però ad avere un notevole traffico di informazioni sul canale trasmissivo, infatti ogni mote deve inviare al PC tutti i valori relativi a tutte le misure effettuate verso tutti i mote. Per ovviare a questo problema il calcolo del modello del canale, eq. 9, viene eseguito utilizzando i valori medi di potenza ricevuta, così ogni mote deve inviare al PC un solo valore, la media della potenza ricevuta, per ogni altro mote presente nel suo raggio d'azione.
Di seguito viene dimostrato che sotto l'ipotesi di aver circa lo stesso numero di misure per ogni distanza il calcolo dei parametri del canale può essere fatto utilizzando la media della potenza ricevuta.
Siano definiti il vettore $\mathbf{b}$ e la matrice A come in 8, è possibile suddividere l'eq. 9 in due parti, ottenendo:

$$(A^T A)^{-1} = \begin{bmatrix} N & -10\sum_{i=1}^{N} \log_{10}(d_i) \\ -10\sum_{i=1}^{N} \log(d_i) & \sum_{i=1}^{N} \left(-10\log_{10} d_i\right)^2 \end{bmatrix}^{-1}$$

$$A^T b = \begin{bmatrix} \sum_{i=1}^{N} P_{rx_i}^{ij} \\ \sum_{i=1}^{N} \left(-10 P_{rx_i}^{ij} \log_{10} d_i\right) \end{bmatrix}$$

dove con d_i si indica la i-esima distanza considerata. Sostituendo N con $\sum_{i=1}^{d} N_i$, $\sum_{i=1}^{N}$ con $\sum_{i=1}^{d}\sum_{k=1}^{N_i}$, indicando con P_{d-k} la k-esima potenza misurata a distanza d ed utilizzando l'ipotesi precedentemente fatta per cui $N_i \simeq N_j = n \;\forall\, i,j = 0,\ldots,d$ si ottiene, dando per implicita la base del logaritmo:

$$(A^T A)^{-1} = \begin{bmatrix} n * d & -10n\sum_{i=1}^{d} \log(d_i) \\ -10n\sum_{i=1}^{d} \log(d_i) & n\sum_{i=1}^{d} \left(-10\log(d_i)\right)^2 \end{bmatrix}^{-1}$$

$$A^T b = \begin{bmatrix} \sum_{i=1}^{d}\sum_{k=1}^{n} P_{i-k} \\ \sum_{i=1}^{d}\sum_{k=1}^{n} \left(-10\log(d_i) P_{i-k}\right) \end{bmatrix}$$

Raccogliendo n da $A^T A$, $\hat{\theta}_{LS}$ diventa:

$$\begin{aligned}\hat{\theta}_{LS} &= \frac{1}{n}\begin{bmatrix} d & -10\sum_{i=1}^{d}\log(d_i) \\ -10\sum_{i=1}^{d}\log(d_i) & \sum_{i=1}^{d}(-10\log(d_i))^2 \end{bmatrix}^{-1}\begin{bmatrix} \sum_{i=1}^{d}\sum_{k=1}^{n} P_{i-k} \\ \sum_{i=1}^{d}\sum_{k=1}^{n}(-10\log(d_i)P_{i-k}) \end{bmatrix} \\ &= \begin{bmatrix} d & -10\sum_{i=1}^{d}\log(d_i) \\ -10\sum_{i=1}^{d}\log(d_i) & \sum_{i=1}^{d}(-10\log(d_i))^2 \end{bmatrix}^{-1}\begin{bmatrix} \frac{1}{n}\sum_{i=1}^{d}\sum_{k=1}^{n} P_{i-k} \\ \frac{1}{n}\sum_{i=1}^{d}\sum_{k=1}^{n}(-10\log(d_i)P_{i-k}) \end{bmatrix} \\ &= \begin{bmatrix} d & -10\sum_{i=1}^{d}\log(d_i) \\ -10\sum_{i=1}^{d}\log(d_i) & \sum_{i=1}^{d}(-10\log(d_i))^2 \end{bmatrix}^{-1}\begin{bmatrix} \sum_{i=1}^{d}\overline{P_i} \\ \sum_{i=1}^{d}(-10\log(d_i)\overline{P_i}) \end{bmatrix}\end{aligned} \tag{10}$$

Ogni nodo quindi misura n valori di potenza per ogni mote nel suo raggio d'azione e ne calcola la media memorizzandola in un array per poi inviarla al PC per il calcolo del modello del canale. I mote non hanno la possibilità di gestire numeri a virgola mobile per cui la media viene calcolata approssimando all'intero più vicino. Questa approssimazione non è limitante, infatti, come precedentemente specificato, l'indice RSSI ha un'accuratezza di 6dBm; molto maggiore dell'approssimazione introdotta.
La procedura di raccolta dati va ripetuta più volte posizionando i sensori a d distanze diverse. Per ognuna di queste il PC memorizza i valori medi di potenza per ogni coppia ordinata di nodi e dopo aver raccolto i dati per le d distanze vengono calcolati i parametri del modello del canale, applicando l'eq. 10, e memorizzati in più file, uno per ogni coppia ordinata di sensori. Questi file vengono utilizzati dal programma che si occupa della raccolta dati per convertire i valori di potenza ricevuti in valori di distanza che vengono utilizzati per stimare le posizioni dei nodi con l'algoritmo di trilaterazione (sez. 2.1).

4.4.2 *Analisi del modello*

Allo scopo di valutare se i sensori in dotazione abbiano caratteristiche che ne permettano l'utilizzo al fine di questo lavoro di tesi si sono effettuate delle prove per verificare se l'errore introdotto nella misura della distanza sia minore dell'errore massimo ammissibile per la convergenza ad una stima corretta dell'algoritmo di trilaterazione.
Il calcolo delle posizioni tramite l'algoritmo di minimizzazione e la memorizzazione dei dati vengono eseguite da un Laptop con: processore *dual core* C-60 a 64 bit con 1 GHz, 4 GB di memoria RAM, 500 GB di hard disk e dotato di 3 porte USB 2.0 su cui è installato Ubuntu 11.10

Sono stati presi 3 mote a campione. Per ogni coppia ordinata di mote sono state considerate d distanze, con d uguale a 4 (nello specifico 1m, 2m, 4m e 6m), e per ognuna di queste sono state effettuare n misurazioni, con n uguale a 8000. Da queste misurazioni si ricavano 6 grafici, Figure 43-48, uno per ogni coppia ordinata di mote, riportanti

un'analisi dei valori registrati; in particolare il grafico relativo alle misurazioni fatte dal nodo i sulla potenza ricevuta dal nodo j riporta: la distribuzione dei valori di potenza misurata dal nodo i sul nodo j in funzione della distanza, il valore medio di potenza misurata per ogni distanza, la curva che descrive il modello del canale ricavata applicando l'eq. 10 e la distanza stimata dal modello del canale partendo dal valore di potenza media misurato.
Inoltre viene riportata una tabella per ogni coppia ordinata di nodi con la distanza reale, la distanza calcolata utilizzando il modello del canale con la potenza media misurata e l'errore commesso nel calcolo della distanza.

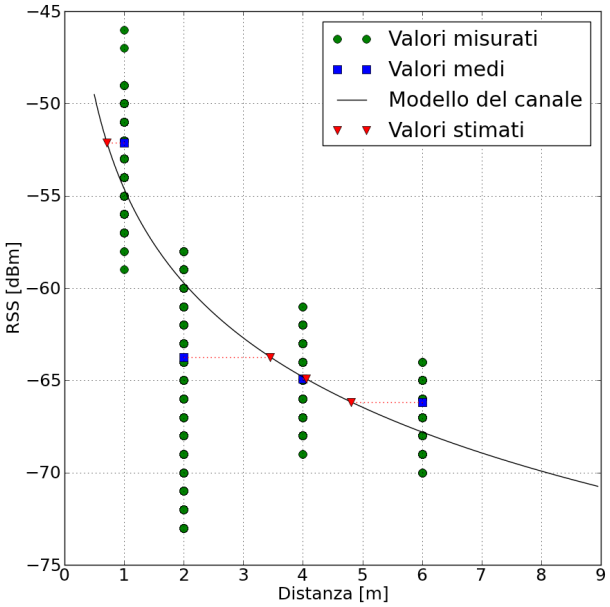


Figura 43: Grafico dei valori di potenza misurati dal nodo 0 rispetto al nodo 1

Distanza reale	Distanza calcolata	Errore commesso
1	0.71	0.29
2	3.45	1.45
4	4.05	0.05
6	4.80	1.2

Tabella 30: Tabella relativa alle misure del nodo 0 rispetto al nodo 1

Dalle tabelle da 30 a 35 si può osservare che il calcolo della distanza attraverso il modello del canale introduce un errore tipicamente maggiore dell'errore massimo ammissibile per la convergenza dell'algoritmo di trilaterazione, cioè $\pm 30\text{cm}$. Errori così elevati nel modello

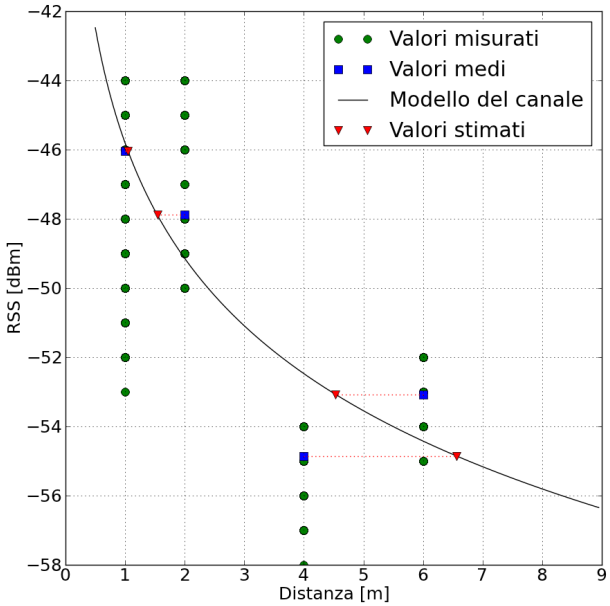


Figura 44: Grafico dei valori di potenza misurati dal nodo 0 rispetto al nodo 2

Distanza reale	Distanza calcolata	Errore commesso
1	1.05	0.05
2	1.54	0.46
4	6.56	2.56
6	4.52	1.48

Tabella 31: Tabella relativa alle misure del nodo 0 rispetto al nodo 2

Distanza reale	Distanza calcolata	Errore commesso
1	0.75	0.25
2	3.15	1.15
4	4.19	0.19
6	4.84	1.16

Tabella 32: Tabella relativa alle misure del nodo 1 rispetto al nodo 0

del canale possono portare, come nel caso di Figura 44, 47, al verificarsi di una sorta di inversione nell'ordine delle distanze infatti, distanze realmente più piccole vengono calcolate, attraverso il modello del canale, maggiori di distanze realmente più grandi.

Si può concludere che questi sensori non sono adatti al loro ruolo all'interno di questo lavoro, cioè misurare le distanze.

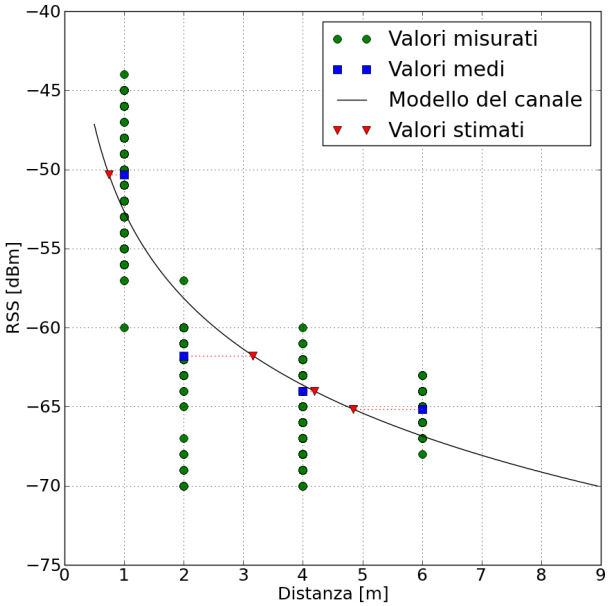


Figura 45: Grafico dei valori di potenza misurati dal nodo 1 rispetto al nodo 0

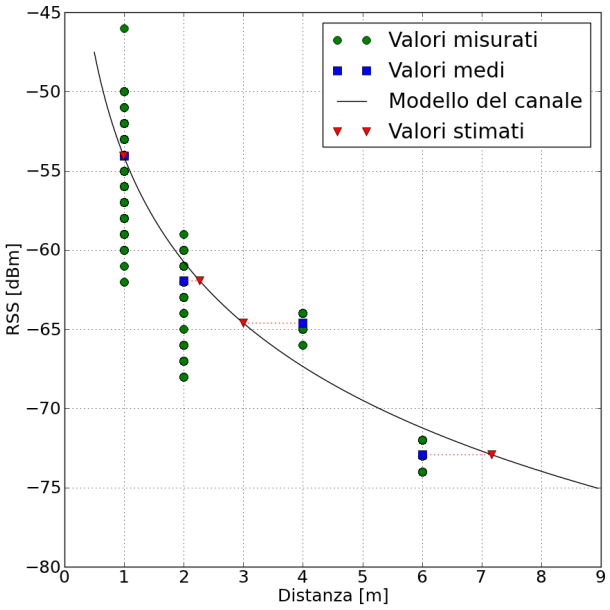


Figura 46: Grafico dei valori di potenza misurati dal nodo 1 rispetto al nodo 2

4.4.3 *Valutazione algoritmo di trilaterazione con dati reali*

Le osservazioni fatte in conclusione alla sezione precedente sono supportate dal risultato ottenuto eseguendo l'algoritmo di trilaterazione

Distanza reale	Distanza calcolata	Errore commesso
1	0.99	0.01
2	2.26	0.26
4	2.99	1.01
6	7.16	1.16

Tabella 33: Tabella relativa alle misure del nodo 1 rispetto al nodo 2

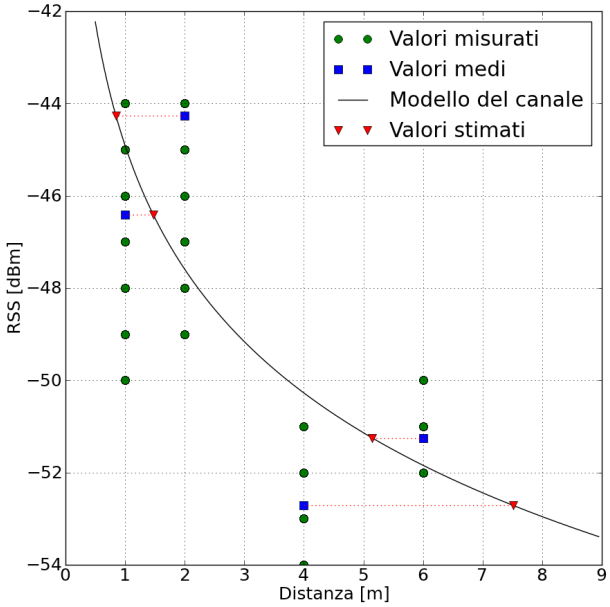


Figura 47: Grafico dei valori di potenza misurati dal nodo 2 rispetto al nodo 0

Distanza reale	Distanza calcolata	Errore commesso
1	1.47	0.47
2	0.84	1.16
4	7.5	3.5
6	5.14	0.86

Tabella 34: Tabella relativa alle misure del nodo 2 rispetto al nodo 0

utilizzando i dati realmente misurati dai sensori. Per ottenere questi risultati si sono fatte 20 misurazioni di distanze e per ogni misurazione si è fatto andare l'algoritmo di trilaterazione utilizzando come condizioni iniziali le posizioni stimate nel caso precedente.
In Figura 49 viene riportata la situazione ottenuta alla fine delle 20 iterazioni. Si può osservare che l'algoritmo non riesce a convergere e viene ottenuto un errore medio di stima, calcolato sulle 20 iterazioni,

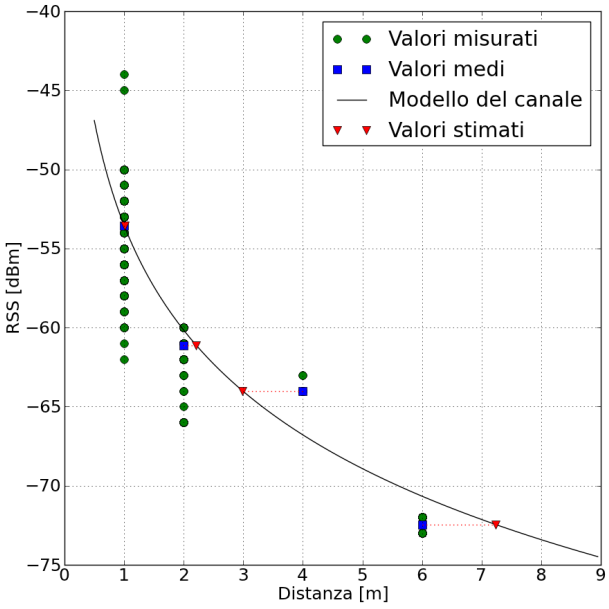


Figura 48: Grafico dei valori di potenza misurati dal nodo 2 rispetto al nodo 1

Distanza reale	Distanza calcolata	Errore commesso
1	1	0
2	2.21	0.21
4	2.99	1.01
6	7.23	1.23

Tabella 35: Tabella relativa alle misure del nodo 2 rispetto al nodo 1

di 2.20m con una deviazione standard di 1m.

Una prova analoga è stata eseguita sui dati di [14], nella quale vengono presentati dei buoni risultati. In questo caso avendo a disposizione un set di dati limitati ad una disposizione dei nodi fissa la localizzazione è stata eseguita una sola volta utilizzando come modello del canale quello calcolato in [14].
In questo caso i risultati ottenuti sono stati buoni, infatti da Figura 50 si può osservare che l'algoritmo riesce ad arrivare a convergenza. Per questa prova l'errore medio è calcolato sull'unica iterazione fatta e risulta essere di 0.24m. Considerando che le distanze in gioco arrivano ad una decina di metri, si può desumere che questo sia un buon risultato.
Tale risultato è supportato anche dall'ottimo esito dell'analisi del modello del canale. A differenza di sez. 4.4.2 in questo caso è stata eseguita un'analisi utilizzando l'unico modello del canale calcolato in [14] valutando i valori di potenza di 4 coppie di mote. In Figura 51 è

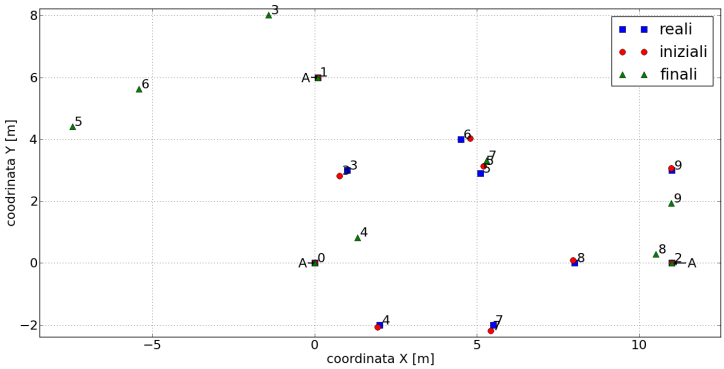


Figura 49: Stima delle posizioni utilizzando i valori di potenza reali.

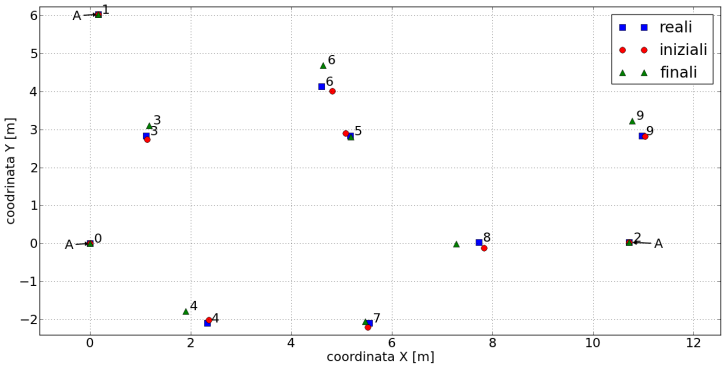


Figura 50: Stima delle posizioni con i valori di potenza utilizzati in [14].

riportato il risultato ottenuto; mentre in Tabella 36 è riportato l'errore di stima del valore della distanza applicando il modello del canale al valore medio di potenza con i dati di [14].

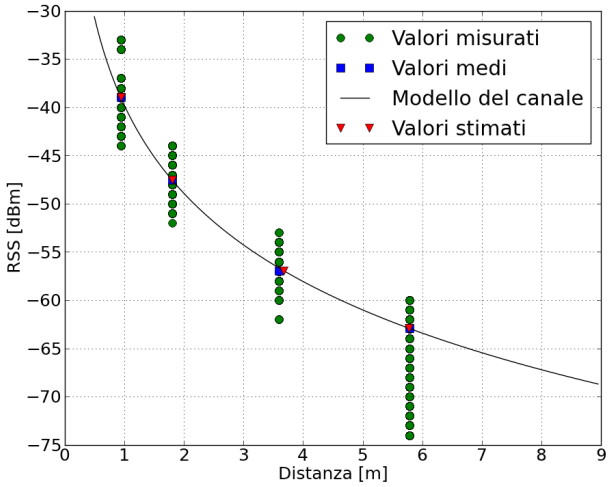


Figura 51: Valutazione del modello del canale con i dati di [14].

Distanza reale	Distanza calcolata	Errore commesso
0.94	0.942	0.002
1.8	1.802	0.002
3.6	3.676	0.076
5.78	5,778	0.002

Tabella 36: Tabella relativa alle misure di [14].

Grazie alle buone prestazioni dei sensori ottenute in [14] questi sensori erano stati impiegati per sviluppare una rete di sensori in grado di localizzarsi.
Tuttavia i risultati sperimentali ottenuti in questo lavoro di tesi portano a concludere che i dispositivi non sono adatti al loro compito all'interno del progetto sviluppato; quindi si può verosimilmente presumere che questi sensori siano affetti da *aging*.

4.5 ACCELEROMETRO

L'accelerometro è un dispositivo in grado di rilevare o misurare un'accelerazione.
Per avere uno schema mentale che ci aiuti a visualizzare come opera un accelerometro, immaginiamolo come costituito da una sfera al centro di un cubo, sospesa da 3 molle che la attraversano, agganciate a loro volta al centro di ogni faccia del cubo; come rappresentato in Figura 52. Muovendo il cubo nello spazio, la sfera si muoverà al suo in-

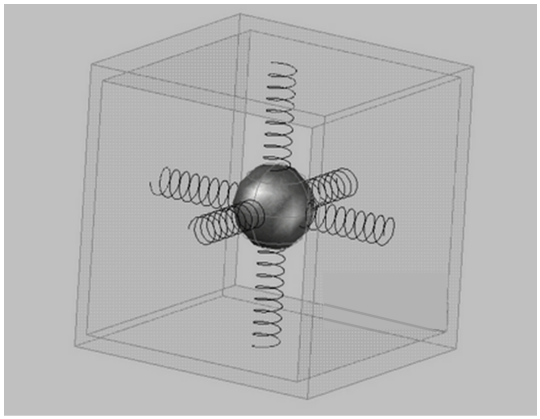


Figura 52: Modello dell'accelerometro

terno allungando e comprimendo, di conseguenza, le molle che la tengono sospesa. La misurazione del grado di compressione delle molle permette di stabilire se c'è stata un'accelerazione nella direzione in cui si trova la molla compressa e quindi anche quantificarla.

L'accelerometro utilizzato è l'ADXL335 della Analog Device [3]: è un accelerometro a 3 assi, piccolo e con bassi consumi di potenza, con un intervallo di misura tipico di $\pm 3.6g$.
Contiene un sensore in poly-silicio ottenuto con lavorazione *surface micromaching*[3] e un circuito per il condizionamento del segnale che permette di implementare un sistema di misura dell'accelerazione ad anello aperto. Il segnale d'uscita è una tensione proporzionale all'accelerazione misurata con condizionamento del segnale di tensione in uscita.
Il chip misura l'accelerazione di gravità, condizione necessaria per applicazioni *tilt-sensing*, e l'accelerazione dinamica dovuta a movimenti, urti e vibrazioni. In Figura 53 viene riportata la misura dell'accelerometro al variare dell'orientazione del chip.
La Figura 54 riporta il diagramma a blocchi funzionale dell'accelerometro, mentre la Tabella37 riporta le specifiche dell'accelerometro.

4.5.1 *Teoria del funzionamento*

Il sensore è una struttura *surface micromachined* costruita sopra un *wafer* di silicio.
Delle molle in poly-silicio tengono la struttura sospesa sopra la superficie del *wafer* e oppongono resistenza alle forze di accelerazione. La

3 Essa consiste nella deposizione di strati dielettrici e metallici sulla superficie di un substrato e la successiva definizione delle strutture tramite tecniche fotolitografiche. La realizzazione di strutture sospese, quali membrane e ponticielli, avviene tramite l'utilizzo di materiali sacrificali. Essi vengono depositati sotto le strutture da rendere sospese e successivamente rimossi tramite tecniche di attacco selettivo, quali il *wet* (in umido) o *dry* (a secco) *reactive-ion etching* (RIE).

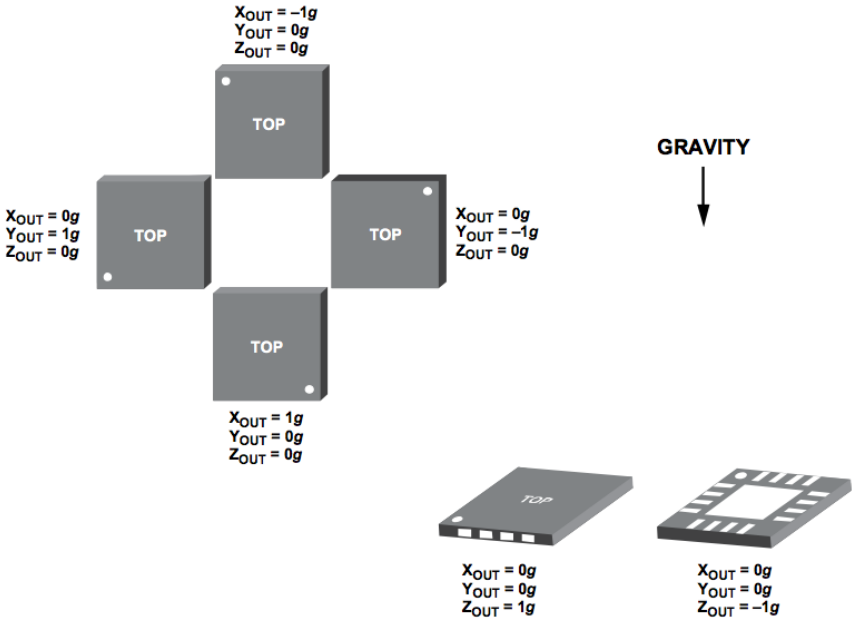


Figura 53: Risposta d'uscita vs. orientamento alla gravità [3].

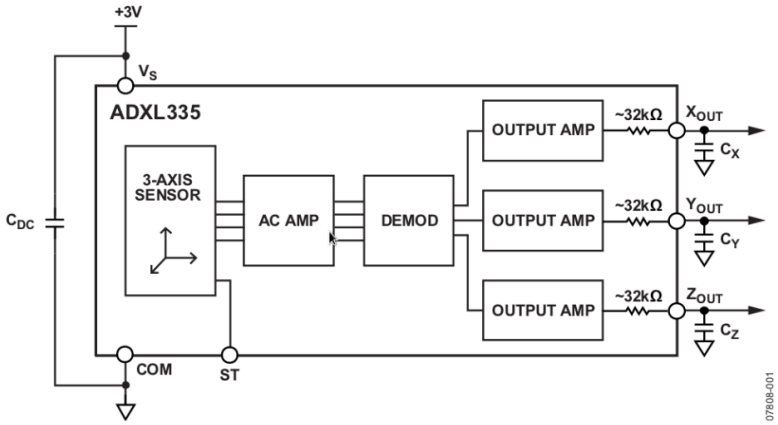


Figura 54: Diagramma a blocchi funzionali dell'accelerometro [3].

Tabella 37: Specifiche dell'accelerometro ADXL335 [3]

Parameter	Conditions	Min	Typ	Max	Unit
SENSOR INPUT	Each axis				
Measurement Range		± 3	± 3.6		g
Nonlinearity	% of full scale		± 0.3		%
Package Alignement Error			± 1		Degree
Interaxis Alignement Error			± 0.1		Degree
Cross-Axis sensitivity			± 1		%
SENSITIVITY	Each axis				
Sensitivity at $X_{OUT}, Y_{OUT}, Z_{OUT}$	$V_s = 3V$	270	300	330	mV/g
sensitivity Change Due to Temperature	$V_s = 3V$		± 0.01		%/°C
ZERO g BIAS LEVEL					
0g Voltage at X_{OUT}, Y_{OUT}	$V_s = 3V$	1.35	1.5	1.65	V
0g Voltage at z_{OUT}	$V_s = 3V$	1.2	1.5	1.8	V
0g Offset vs. Temperature			± 1		$mg/°C$
NOISE PERFORMANCE					
Noise Density X_{OUT}, Y_{OUT}			150		$\mu g/\sqrt{Hz}\, rms$
Noise Density Z_{OUT}			300		$\mu g/\sqrt{Hz}\, rms$
FREQUENCY RESPONSE					
Bandwidth X_{OUT}, Y_{OUT}	No external filter		1600		Hz
Bandwidth Z_{OUT}	No external filter		550		Hz
R_{FILT} Tolerance			$32 \pm 15\%$		$k\Omega$
Sensor Resonat Frequency			5.5		kHz
SELF-TEST					
Logic Input Low			$+0.6$		V
Logic Input High			$+2.4$		V
ST Actuation Current			$+60$		μA
Output Change at X_{OUT}	Self-Test 0 to Self-Test 1	-150	-325	-600	mV
Output Change at Y_{OUT}	Self-Test 0 to Self-Test 1	$+150$	$+325$	$+600$	mV
Output Change at Z_{OUT}	Self-Test 0 to Self-Test 1	$+150$	$+550$	$+1000$	mV
OUTPUT AMPLIFIER					
Output Swing Low	No load			0.1	V
Output Swing High	No load			2.8	V
POWER SUPPLY					
Operating Voltage Range		1.8		3.6	V
Supply Current	$V_s = 3V$		350		μA
Turn-On Time	No external filter		1		ms
TEMPERATURE					
Operating Temperature Range		-40		$+85$	°C

deformazione della struttura viene misurata attraverso un condensatore differenziale, che consiste in due piastre indipendenti, una fissa e l'altra collegata alla massa in movimento. La piastra fissa viene alimentata con un onda quadra. L'accelerazione devia la massa in movimento e sbilancia il condensatore differenziale producendo un'uscita la cui ampiezza è proporzionale all'accelerazione. Tecniche di demodulazione *phase-sensitive* sono poi usate per determinare l'ampiezza e la direzione dell'accelerazione.
L'uscita del demodulatore è amplificata e viene portata fuori dal chip attraverso un resistore da 32kΩ.
È possibile scegliere la larghezza di banda dell'accelerometro aggiungendo dei condensatori all'uscita del chip. Questo filtraggio migliora la risoluzione di misura e aiuta a prevenire l'*aliasing*.

L'ADXL335 utilizza una singola struttura per misurare l'accelerazione sui 3 assi. Come conseguenza si ha una precisa l'ortogonalità degli assi e una bassa sensibilità inter-asse.
La sensibilità inter-asse presente è dovuta al disalineamento meccanico del sensore rispetto al *package* del chip; questo disalineamento può, ovviamente, essere calibrato a livello di sistema.

Le innovative tecniche di costruzione utilizzate per questo chip garantiscono delle buone prestazioni anche senza un circuito addizionale per la compensazione delle variazioni di temperature. Infatti non ci sono errori di quantizzazione o comportamenti non-monotoni, e l'isteresi dovuta alla temperaturea è molto bassa (tipicamente meno di 3mg su un intervallo di temperatura da $-25°C$ e $+70°C$).

4.5.2 *Informazioni sull'utilizzo*

Come si vede da Figura 54 è presente un condensatore, C_{DC}, da 0.1μF, collegato tra i pin di alimentazione. Questo è sufficiente nella maggior parte delle applicazioni per disaccoppiare l'alimentazione dai disturbi. Tuttavia, se il disturbo è ad una frequenza di 50Hz come il clock interno è necessario adottare particolari tecniche per bypassare il problema, perchè questo disturbo può causare degli errori sulla misurazione dell'accelerazione.
Per ottenere un disaccoppiamento supplementare è possibile inserire un resistore da 100Ω (o più piccolo) nella linea di alimentazione e/o un condensatore da 1μF (o maggiore) in parallelo a C_{CD}.

Collegando dei condensatori ai pin X_{OUT}, Y_{OUT} e Z_{OUT} si implementa un filtro passa-basso per la riduzione dei disturbi e l'*antialiasing*. L'equazione per il calcolo della frequenza di taglio è riportata in 11.

$$F_{-3dB} = \frac{1}{2\pi(32k\Omega) \times C_{(X,Y,Z)}} \tag{11}$$

La tolleranza della resistenza interna (R_{FILT}) tipicamente varia di ±15% rispetto al valore nominale, come riportato in Tabella37, e la banda passante varia di conseguenza.

La larghezza di banda dell'accelerometro determina in ultima analisi la risoluzione di misura; questa può essere migliorata abbassando il livello di rumore attraverso un filtraggio.
L'uscita dell'ADXL335 ha una larghezza di banda tipicamente maggione di 500Hz. La banda analogica non deve essere maggiore della metà della frequenza di campionamento del convertitore analogico-digitale per minimizzare l'*aliasing*.
Il rumore del sensore ha le caratteristiche di un rumore bianco gaussiano, che contribuisce in modo uniforme a tutte le frequenze ed è descritto in $\mu g/\sqrt{Hz}$. Bisogna limitare la banda alla frequenza più bassa necessaria per l'applicazione in modo da massimizzare la risoluzione e il range dinamico dell'accelerometro.

Il pin ST controlla il *self-test*; questa caratteristica consente di verificare il funzionamento dell'accelerometro. Quando questo pin è impostato su V_s viene esercitata una forza elettrostatica che produce delle variazioni delle uscite. Variazioni tipiche sono riportate in Tabella37.

Il chip viene testato con una tensione di alimentazione, V_s, di 3V; tuttavia può essere alimentato con tensioni comprese tra 1.8V e 3.6V. Si deve però fare attenzione che al variare della tensione di alimentazione variano anche alcuni parametri del sensore. L'uscita del ADXL335 infatti è raziometrica, per cui, il suo zero e la sua sensibilità dipendono dalla tensione di alimentazione.
Sia la sensibilità che lo zero variano in modo proporzionale alla tensione, infatti, la sensibilità è uguale a circa un decimo della tensione di alimentazione (a $V_s = 3.6V$ la sensibilità d'uscita è $360mV/g$, a $V_s = 2V$ è $195mV/g$) e l'uscita per 0g è di $V_s/2$.
Il rumore d'uscita, al contario, non è raziometrico ma assume un valore fisso in V: pertanto, la densità di rumore decresce al crescere della tensione d'alimetazione. Questo avviene perchè il fattore di scala (mV/g) cresce mentre il rumore in volts rimane costante. A $V_s = 3.6V$ la densità di rumore per gli assi X e Y è tipicamente di $120\mu g/\sqrt{Hz}$, mentre con $V_s = 2V$ la densità di rumore sugli stessi assi è di $270\mu g/\sqrt{Hz}$.
Altri due parametri che dipendono dalla tensione di alimentazione sono: la risposta al self-test che in volt è circa proporzionale al cubo della tensione (per esempio con alimentazione a $3,6V$ la risposta del self-test è per gli assi X e Y circa $560mV$, mentre con alimentazione a 2V la risposta per gli stessi assi è di circa $96mV$) ed il consumo di corrente che decresce al diminuire della tensione di alimentazione.

4.6 VERIFICA FUNZIONAMENTO ACCELEROMETRO

In questa fare iniziale dello sviluppo è stato acquistato un sono accelerometro per verificarne il funzionamento e l'integrazione con il mote; la possibilità di acquistare accelerometri in numero sufficiente da equipaggiare tutti i nodi verrà vagliata in seguito.

L'accelerometro viene collegato al connettore di espansione U2 del Tmote; da questo prende l'alimentazione e utilizza gli ADC per fornire i valori di accelerazione dei tre assi.

Il valore analogico di tensione, proporzionale all'accelerazione misurata, è fortemente dipendente dalla tensione di alimentazione; per questo la conversione dal valore digitale ottenuto dall'ADC al valore di gravità viene effettuata tenendo conto del livello di tensione delle batterie che alimentano il mote. Vista l'impossibilità dei mote di effettuare calcoli in virgola mobile questa conversione viene effettuata su PC.

Per la gestione degli ADC sono stati creati tre componenti, uno per ogni asse dell'accelerometro.
Ogni componente fornisce un interfaccia per richiedere la lettura del valore convertito dall'ADC e configura l'ADC stesso.
La configurazione consiste nel settaggio della porta di input, la tensione di riferimento per la conversione e dei tempi di *sample & hold*.

Per visualizzare le misure eseguite dall'accelerometro è stato utilizzato un programma fornito nel pacchetto di installazione di TinyOS. In Figura 55 viene riportato uno *screenshot* di questo programma. Per ottenere questo grafico si è fatto ruotare il mote di 180° rispetto ad uno degli assi; infatti le componenti della forza di gravità rimangono costanti lungo due assi mentre lungo il terzo cambia segno.

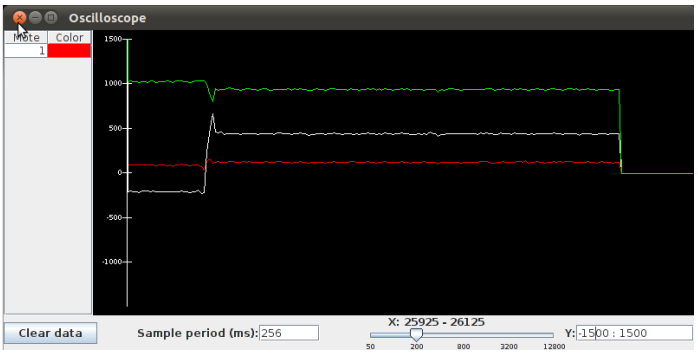


Figura 55: Screenshot programma java per visualizzare i valori misurati dall'accelerometro.

CONCLUSIONI

In questa tesi è stato presentato un metodo per il monitoraggio di eventi di colata detritica attraverso lo sviluppo di una rete di sensori wireless, ancorati a detriti, che si autolocalizzano.

Nel secondo capitolo è stata illustrata la soluzione proposta descrivendo l'algoritmo di trilaterazione utilizzato per effettuare la stima della posizione dei nodi a partire dalle distanze misurate da essi e il suo sviluppo implementativo in C++ per permetterne l'utilizzo all'interno del progetto finale.

Il terzo capitolo riporta le analisi effettuare per quantificare la precisione dell'algoritmo di trilaterazione proposto al variare dell'errore introdotto nella misura delle distanze. Tali analisi sono state effettuate sia in condizioni statiche, quindi considerando i mote fermi, sia in condizioni dinamiche, quindi considerando l'eventualità che si possa verificare uno spostamento di un nodo. Le simulazioni svolte dimostrano la possibilità di avere un buon funzionamento dell'algoritmo di trilaterazione se le distanze misurate hanno un errore di misura con una deviazione standard minore di 10cm. Inoltre in questa condizione si è concluso che la disposizione dei nodi non influenza in modo rilevante la stima delle posizioni infatti le due disposizioni dei nodi considerate restituiscono un errore di stima sostanzialmente uguale. Diverso è invece il comportamento dell'algoritmo all'aumentare dei nodi ancòra. In questo caso, essendo la rete di nodi più vincolata, l'algoritmo di trilaterazione restituendo una stima delle posizione dei mote migliore. Tale miglioramento è però presente solo se l'errore di distanza introdotto non è elevato, infatti se l'errore introdotto ha una deviazione standard maggiore di 20cm l'aggiunta di un'ancòra non porta nessun tipo di vantaggio.

Il quarto capitolo descrive in dettaglio le caratteristiche hardware e software del dispositivo scelto per lo sviluppo del progetto e dell'accelerometro triassale scelto per essere montato sul sensore al fine di ottenere una misura più precisa degli spostamenti dovuti ad eventuali colate detritiche. Il capitolo presenta una descrizione del modello di canale utilizzato e i risultati ottenuti con i dati raccolti su una serie di sensori per valutare se questo sensore permette un errore nel calcolo delle distanze inferiore a quello richiesto dall'algoritmo di trilaterazione. L'analisi fatta sui dati raccolti dai sensori ha portato a concludere che tali dispositivi non sono in grado di misurare la distanza con un errore entro i limiti imposti dall'algoritmo di trilaterazione; di conseguenza i sensori non sono adatti per misurare la distanza, dato di input per l'algoritmo di trilaterazione. Dati raccolti precedentemente sullo stesso tipo di sensori, affetti da un errore com-

patibile con i vincoli dell'algoritmo di trilaterazione, dimostrano la bontà dell'algoritmo proposto che riesce a convergere e a localizzare i sensori.

Negli sviluppi futuri del progetto è previsto l'acquisto di altri sensori con caratteristiche migliori per poter essere utilizzati all'interno del progetto. Altri possibili sviluppi riguardano l'accelerometro. Il dispositivo scelto è compatibile con le specifiche del problema e si prevede l'acquisto e l'integrazione sui nuovi sensori. Questo permetterà di sviluppare nuovi algoritmi che, a partire dai valori ricevuti dall'accelerometro, siano in grado di individuare possibili situazioni di pericolo che richiedono un monitoraggio più frequente della zona. Un ulteriore strada su cui concentrare gli sviluppi riguarda la creazione di un server web che mostri i dati rilevati attraverso una interfaccia utente studiata per mostrare sia i dati correnti che lo storico degli spostamenti.

APPENDIX

A

SETUP DEL SOFTWARE

In questa appendice vengono riportati i passaggi per installare il software TinyOS 2.1.0 sul proprio PC con Ubuntu 11.10.
Il manuale di installazione online del TinyOS 2.1, reperibile all'indirizzo *http://docs.tinyos.net/index.php/Installing_TinyOS_2.1*, è riferito a versioni di Ubuntu più vecchie di quella presente sul PC utilizzato per lo sviluppo. Per tale versione, 11.10, è stato necessario eseguire i seguenti passaggi.

A.1 PACCHETTI UBUNTU

Prima di installare TinyOS di deve aggiungere il *repository* di TinyOS nel *repository source file*. Eseguendo il seguente comando

```
$ sudo gedit /etc/apt/sources.list
```

viene aperto il file *sources.list* in una finestra di dialogo come il Figura 56. Dopo aver aggiunto a questo file le seguenti righe di configurazio-

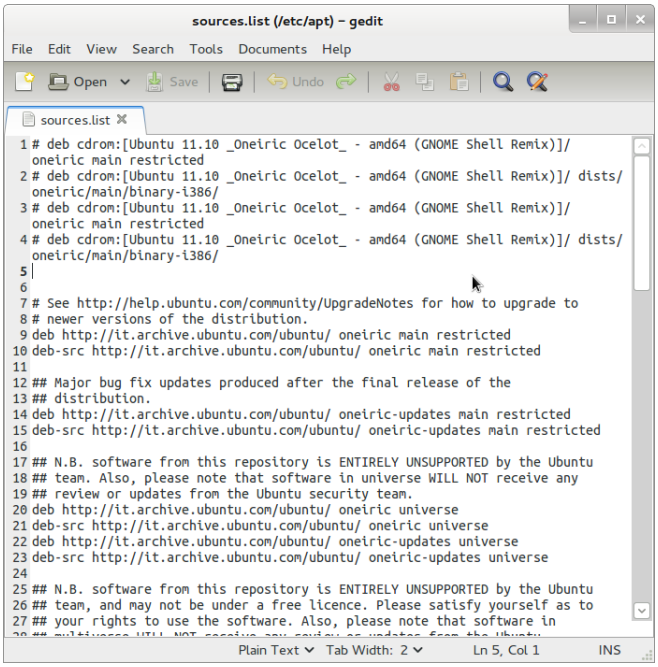

sources.list (/etc/apt) – gedit
File Edit View Search Tools Documents Help
Open Save Undo
sources.list
1 # deb cdrom:[Ubuntu 11.10 _Oneiric Ocelot_ - amd64 (GNOME Shell Remix)]/ oneiric main restricted
2 # deb cdrom:[Ubuntu 11.10 _Oneiric Ocelot_ - amd64 (GNOME Shell Remix)]/ dists/ oneiric/main/binary-i386/
3 # deb cdrom:[Ubuntu 11.10 _Oneiric Ocelot_ - amd64 (GNOME Shell Remix)]/ oneiric main restricted
4 # deb cdrom:[Ubuntu 11.10 _Oneiric Ocelot_ - amd64 (GNOME Shell Remix)]/ dists/ oneiric/main/binary-i386/
5
6
7 # See http://help.ubuntu.com/community/UpgradeNotes for how to upgrade to
8 # newer versions of the distribution.
9 deb http://it.archive.ubuntu.com/ubuntu/ oneiric main restricted
10 deb-src http://it.archive.ubuntu.com/ubuntu/ oneiric main restricted
11
12 ## Major bug fix updates produced after the final release of the
13 ## distribution.
14 deb http://it.archive.ubuntu.com/ubuntu/ oneiric-updates main restricted
15 deb-src http://it.archive.ubuntu.com/ubuntu/ oneiric-updates main restricted
16
17 ## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
18 ## team. Also, please note that software in universe WILL NOT receive any
19 ## review or updates from the Ubuntu security team.
20 deb http://it.archive.ubuntu.com/ubuntu/ oneiric universe
21 deb-src http://it.archive.ubuntu.com/ubuntu/ oneiric universe
22 deb http://it.archive.ubuntu.com/ubuntu/ oneiric-updates universe
23 deb-src http://it.archive.ubuntu.com/ubuntu/ oneiric-updates universe
24
25 ## N.B. software from this repository is ENTIRELY UNSUPPORTED by the Ubuntu
26 ## team, and may not be under a free licence. Please satisfy yourself as to
27 ## your rights to use the software. Also, please note that software in
Plain Text Tab Width: 2 Ln 5, Col 1 INS

Figura 56: *Repository source file*

ne:

```
 #TinyOS
deb http://hinrg.cs.jhu.edu/tinyos karmic main
```

salvare e chiudere il file.

A.1.1 *Installazione TinyOS*

In primo luogo si deve aggiornare il *repository* con il comando

```
$ sudo apt-get update
```

successivamente si può procedere all'installazione di TinyOS 2.1.1 utilizzando il comando

```
$ sudo apt-get install tinyos-2.1.1
```

Dopo l'installazione è necessario aggiornale il file *bashrc*. Si utilizza il comando

```
$ gedit ~/.bashrc
```

per aprire il file, Figura 57, al quale vanno aggiunte le seguenti righe:

```
export TOSROOT=/opt/tinyos-2.1.1

export TOSDIR=$TOSROOT/tos

export CLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:.:
    $CLASSPATH

export MAKERULES=$TOSROOT/support/make/Makerules

export PATH=/opt/msp430/bin:$PATH

#Sourcing the tinyos environment variable setup script

source /opt/tinyos-2.1.1/tinyos.sh
```

A.1.2 *Verifica dell'installazione*

Per verificare se l'installazione è andata a buon fine si esegue il comando

```
$ tos-check-env
```

questo verifica se il sistema è compatibile con l'utilizzo del software TinyOS e restituisce come output sul terminale i percorsi contenuti nelle variabili ambiente *path* e *classpath* e le versioni dei compilatori di nesC, perl, Java e graphviz; dando errore se qualcuna di queste versioni non è aggiornata per TinyOS.

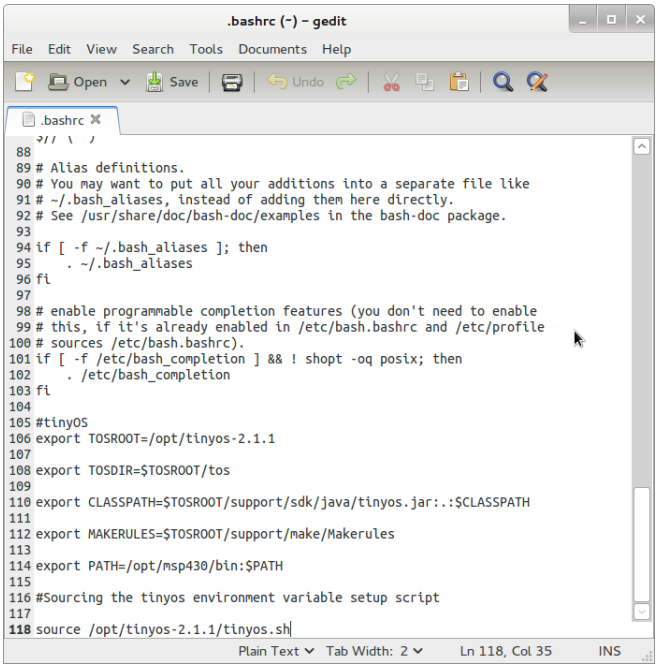

.bashrc (~) – gedit
File Edit View Search Tools Documents Help
Open Save Undo
.bashrc
88
89 # Alias definitions.
90 # You may want to put all your additions into a separate file like
91 # ~/.bash_aliases, instead of adding them here directly.
92 # See /usr/share/doc/bash-doc/examples in the bash-doc package.
93
94 if [-f ~/.bash_aliases]; then
95 . ~/.bash_aliases
96 fi
97
98 # enable programmable completion features (you don't need to enable
99 # this, if it's already enabled in /etc/bash.bashrc and /etc/profile
100 # sources /etc/bash.bashrc).
101 if [-f /etc/bash_completion] && ! shopt -oq posix; then
102 . /etc/bash_completion
103 fi
104
105 #tinyOS
106 export TOSROOT=/opt/tinyos-2.1.1
107
108 export TOSDIR=$TOSROOT/tos
109
110 export CLASSPATH=$TOSROOT/support/sdk/java/tinyos.jar:.:$CLASSPATH
111
112 export MAKERULES=$TOSROOT/support/make/Makerules
113
114 export PATH=/opt/msp430/bin:$PATH
115
116 #Sourcing the tinyos environment variable setup script
117
118 source /opt/tinyos-2.1.1/tinyos.sh
Plain Text Tab Width: 2 Ln 118, Col 35 INS

Figura 57: File bashrc aggiornato

BIBLIOGRAFIA

[1] URL http://www.sentilla.com.

[2] M. Vanin A. Martini, A. Preti. *Localization and Tracking in Wireless Sensor Networks*. PhD thesis, Università degli studi di Padova. Dipartimento di ingegneria dell'informazione., A.A. 2011-2012.

[3] *Datasheet: Small, Low power, 3-Axis ± 3 g Accelerometer - ADXL335*. Analog Device, One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A., rev. b edition. URL http://www.analog.com.

[4] Lafuente Marienela Carlos Genatios. Lluvias torrenciales en vargas, venezuela, en diciembre de 1999. protección ambiental y recuperación urbana. 41(2-3):49–62, 2003.

[5] M. Srivastava D. Culler, D. Estrin. Overview of sensors networks. *IEEE Computer Society.*, 37:41–49, August 2004.

[6] Texas Instruments. *MSP430x1xx Family User's Guide*. URL http://ti.com/msp430.

[7] O. Tingleff K. Madsen, H.B. Nielsen. *Method for non-linear least square problems*. PhD thesis, Technical University of Denmark, April 2004.

[8] moteIV. Tmote sky. Datasheet, 6 2006.

[9] L. Parolini. *Metodi di localizzazione per reti di sensori wireless*. PhD thesis, Università degli studi di Padova. Dipartimento di ingegneria dell'informazione, A.A. 2005-2006.

[10] R.S. Pressman. *Principi di ingegneria del software 4ed*. McGraw-Hill, 2005.

[11] Chipcon Products. *CC2420 2.4GHz IEEE 802.15.4 ZigBee-ready RF Transceiver*. Texas Instruments, Feb 2012. URL http://www.ti.com.

[12] G. Pucci. Metodi statistici per l'identificazione di sistemi lineari. Dipartimento di Ingegneria dell' Informazione, Universita' di Padova, Gennaio 2011.

[13] IEEE Standards. *802.15.4e-2012 - IEEE Standard for Local and metropolitan area networks– Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs) Amendment 1 : MAC sublayer*, 2012. URL http://standards.ieee.org/findstds/standard/802.15.4e-2012.html.

[14] F. Zanella. *Localizzazione e tracking multioggetto per reti di sensori wireless*. PhD thesis, Università degli studi di Padova. Dipartimento di ingegneria dell'informazione., 2008.